# reader Documentation

***Release 3.12***

**lemon24**

**Mar 05, 2024**

# CONTENTS

**reader** is a Python feed reader library.

It is designed to allow writing feed reader applications without any business code, and without depending on a particular framework.

CHAPTER

ONE

# FEATURES

*reader* allows you to:

- retrieve, store, and manage **Atom**, **RSS**, and **JSON** feeds
- mark articles as read or important
- add arbitrary tags/metadata to feeds and articles
- filter feeds and articles
- full-text search articles
- get statistics on feed and user activity
- write plugins to extend its functionality
- skip all the low level stuff and focus on what makes your feed reader different

. . . all these with:

- a stable, clearly documented API
- excellent test coverage
- fully typed Python

What *reader* doesn't do:

- provide an UI
- provide a REST API (yet)
- depend on a web framework
- have an opinion of how/where you use it

The following exist, but are optional (and frankly, a bit unpolished):

- a minimal web interface
  - that works even with text-only browsers
  - with automatic tag fixing for podcasts (MP3 enclosures)
- a command-line interface

CHAPTER

TWO

# QUICKSTART

What does it look like? Here is an example of *reader* in use:

```
$ pip install reader
```

```
>>> from reader import make_reader
>>>
>>> reader = make_reader('db.sqlite')
>>> reader.add_feed('http://www.hellointernet.fm/podcast?format=rss')
>>> reader.update_feeds()
>>>
>>> entries = list(reader.get_entries())
>>> [e.title for e in entries]
['H.I. #108: Project Cyclops', 'H.I. #107: One Year of Weird', ...]
>>>
>>> reader.mark_entry_as_read(entries[0])
>>>
>>> [e.title for e in reader.get_entries(read=False)]
['H.I. #107: One Year of Weird', 'H.I. #106: Water on Mars', ...]
>>> [e.title for e in reader.get_entries(read=True)]
['H.I. #108: Project Cyclops']
>>>
>>> reader.update_search()
>>>
>>> for e in reader.search_entries('year', limit=3):
...     title = e.metadata.get('.title')
...     print(title.value, title.highlights)
...
H.I. #107: One Year of Weird (slice(15, 19, None),)
H.I. #52: 20,000 Years of Torment (slice(17, 22, None),)
H.I. #83: The Best Kind of Prison ()
```

CHAPTER

THREE

# USER GUIDE

This part of the documentation guides you through all of the library's usage patterns.

## 3.1 Why *reader*?

### 3.1.1 Why use a feed reader library?

Have you been unhappy with existing feed readers and wanted to make your own, but:

- never knew where to start?
- it seemed like too much work?
- you don't like writing backend code?

Are you already working with feedparser, but:

- want an easier way to store, filter, sort and search feeds and entries?
- want to get back type-annotated objects instead of dicts?
- want to restrict or deny file-system access?
- want to change the way feeds are retrieved by using the more familiar requests library?
- want to also support JSON Feed?
- want to support custom information sources?

… while still supporting all the feed types feedparser does?

If you answered yes to any of the above, *reader* can help.

### 3.1.2 The *reader* philosophy

- *reader* is a library
- *reader* is for the long term
- *reader* is extensible
- *reader* is stable (within reason)
- *reader* is simple to use; API matters
- *reader* features work well together
- *reader* is tested

- *reader* is documented
- *reader* has minimal dependencies

### 3.1.3 Why make your own feed reader?

So you can:

- have full control over your data
- control what features it has or doesn't have
- decide how much you pay for it
- make sure it doesn't get closed while you're still using it
- really, it's easier than you think

Obviously, this may not be your cup of tea, but if it is, *reader* can help.

### 3.1.4 Why make a feed reader library?

I wanted a feed reader that is:

- accessible from multiple devices
- fast
- with a simple UI
- self-hosted (for privacy reasons)
- modular / easy to extend (so I can change stuff I don't like)
- written in Python (see above)

The fact that I couldn't find one extensible enough bugged me so much that I decided to make my own; a few years later, I ended up with what I would've liked to use when I first started.

## 3.2 Installation

### 3.2.1 Python versions

*reader* supports Python 3.10 and newer, and PyPy.

### 3.2.2 Dependencies

These packages will be installed automatically when installing *reader*:

- feedparser parses feeds; *reader* is essentially feedparser + state.
- requests retrieves feeds from the internet; it replaces feedparser's default use of `urllib` to make it easier to write plugins.
- werkzeug provides HTTP utilities.
- iso8601 parses dates in ISO 8601 / RFC 3339; used for JSON Feed parsing.
- beautifulsoup4 is used to strip HTML tags before adding entries to the search index.

- typing-extensions is used for `typing` backports.

*reader* also depends on the `sqlite3` standard library module (at least SQLite 3.15), and on the JSON1 SQLite extension. To use the *full-text search* functionality, at least SQLite 3.18 with the FTS5 extension is required.

#### Optional dependencies

Despite coming with a CLI and web application, *reader* is primarily a library. As such, most dependencies are optional, and can be installed as extras.

As of version 3.12, *reader* has the following extras:

- `cli` installs the dependencies needed for the *command-line interface*.
- `app` installs the dependencies needed for the *web application*.
- Specific plugins may require additional dependencies; refer to their documentation for details.

### 3.2.3 Virtual environments

You should probably install *reader* inside a virtual environment; see this for how and why to do it.

### 3.2.4 Install reader

Use the following command to install *reader*, along with its required dependencies:

```
pip install reader
```

Use the following command to install *reader* with *optional dependencies*:

```
pip install 'reader[some-extra,...]'
```

#### Update reader

Use the following command to update *reader* (add any extras as needed):

```
pip install --upgrade reader
```

#### Living on the edge

If you want to use the latest *reader* code before it's released, install or update from the master branch:

```
pip install --upgrade https://github.com/lemon24/reader/archive/master.tar.gz
```

## 3.3 Tutorial

In this tutorial we'll use *reader* to download all the episodes of a podcast, and then each new episode as they come up.

Podcasts are episodic series that share information as digital audio files that a user can download to a personal device for easy listening. Usually, the user is notified of new episodes by periodically downloading an RSS feed which contains links to the actual audio files; in the context of a feed, these files are called *enclosures*.

The final script is available as an example in the *reader* repository, if you want to compare your script with the final product as you follow the tutorial.

**Note:** Before starting, install *reader* by following the instructions *here*.

### 3.3.1 Adding and updating feeds

Create a `podcast.py` file:

```
from reader import make_reader

feed_url = "http://www.hellointernet.fm/podcast?format=rss"

reader = make_reader("db.sqlite")

def add_and_update_feed():
    reader.add_feed(feed_url, exist_ok=True)
    reader.update_feeds()

add_and_update_feed()

feed = reader.get_feed(feed_url)
print(f"updated {feed.title} (last changed at {feed.updated})\n")
```

*make_reader()* creates a *Reader* object; this gives access to most *reader* functionality and persists the state related to feeds to a file.

*add_feed()* adds a new feed to the list of feeds. Since we will run the script repeatedly to download new episodes, if the feed already exists, we can just move along.

*update_feeds()* retrieves and stores all the added feeds.

*get_feed()* returns a *Feed* object that contains information about the feed. We could have called *get_feed()* before *update_feeds()*, but the returned feed would have most of its attributes set to None, which is not very useful.

Run the script with the following command:

```
python3 podcast.py
```

The output should be similar to this:

```
updated Hello Internet (last changed at 2020-02-28 09:34:02+00:00)
```

Comment out the `add_and_update_feed()` call for now. If you re-run the script, the output should be the same, since *get_feed()* returns data already persisted in the database.

### 3.3.2 Looking at entries

Let's look at the individual elements in the feed (called *entries*); add this to the script:

```
def download_everything():
    entries = reader.get_entries()
    entries = list(entries)[:3]

    for entry in entries:
        print(entry.feed.title, '-', entry.title)

download_everything()
```

By default, *get_entries()* returns an iterable of all the entries of all the feeds, most recent first.

In order to keep the output short, we only look at the first 3 entries for now. Running the script should output something like this (skipping that first "updated ..." line):

```
Hello Internet - H.I. #136: Dog Bingo
Hello Internet - H.I. #135: Place Your Bets
Hello Internet - # H.I. 134: Boxing Day
```

At the moment we only have a single feed; we can make sure we only get the entries for this feed by using the *feed* argument; while we're at it, let's also only get the entries that have enclosures:

```
entries = reader.get_entries(feed=feed_url, has_enclosures=True)
```

Note that we could have also used `feed=feed`; wherever Reader needs a feed, you can pass either the feed URL or a *Feed* object. This is similar for entries; they are identified by a (feed URL, entry id) tuple, but you can also use an *Entry* object instead.

### 3.3.3 Reading entries

As mentioned in the beginning, the script will keep track of what episodes it already downloaded and only download the new ones.

We can achieve this by getting the unread entries, and marking them as read after we process them:

```
entries = reader.get_entries(feed=feed_url, has_enclosures=True, read=False)
...

for entry in entries:
    ...
    reader.mark_entry_as_read(entry)
```

If you run the script once, it should have the same output as before. If you run it again, it will show the next 3 unread entries:

```
Hello Internet - Star Wars: The Rise of Skywalker, Hello Internet Christmas Special
Hello Internet - H.I. #132: Artisan Water
Hello Internet - H.I. #131: Panda Park
```

### 3.3.4 Downloading enclosures

Once we have the machinery to go through entries in place, we can move on to downloading enclosures.

First we add some imports we'll use later, and a variable for the path of the download directory:

```
import os
import os.path
...
podcasts_dir = "podcasts"
```

In order to make testing easier, we initially write a dummy download_file() function that only writes the enclosure URL to the file instead of downloading it:

```
def download_file(src_url, dst_path):
    with open(dst_path, 'w') as file:
        file.write(src_url + '\n')
```

And then we use it in download_everything():

```
for entry in entries:
    print(entry.feed.title, '-', entry.title)

    for enclosure in entry.enclosures:
        filename = enclosure.href.rpartition('/')[2]
        print("  *", filename)
        download_file(enclosure.href, os.path.join(podcasts_dir, filename))

    reader.mark_entry_as_read(entry)
```

For each *Enclosure*, we extract the filename from the enclosure URL so we can use it as the name of the local file.

*mark_entry_as_read()* gets called *after* we download the file, so if the download fails, the script won't skip it at the next re-run.

We also need to make sure the directory exists before calling download_everything(), otherwise trying to open a file in it will fail:

```
os.makedirs(podcasts_dir, exist_ok=True)
download_everything()
```

Running the script now should create three .mp3 files in *podcasts/*:

```
Hello Internet - H.I. #130: Remember Harder
  * 130.mp3
Hello Internet - H.I. #129: Sunday Spreadsheets
  * 129.mp3
Hello Internet - H.I. #128: Complaint Tablet Podcast
  * 128.mp3
```

```
$ for file in podcasts/*; do echo '#' $file; cat $file; done
# podcasts/128.mp3
http://traffic.libsyn.com/hellointernet/128.mp3
# podcasts/129.mp3
http://traffic.libsyn.com/hellointernet/129.mp3
```

(continues on next page)

(continued from previous page)

```
# podcasts/130.mp3
http://traffic.libsyn.com/hellointernet/130.mp3
```

With everything wired up correctly, we finally implement the download function using `requests`:

```
import shutil
import requests

...

def download_file(src_url, dst_path):
    part_path = dst_path + '.part'
    with requests.get(src_url, stream=True) as response:
        response.raise_for_status()
        try:
            with open(part_path, 'wb') as file:
                shutil.copyfileobj(response.raw, file)
            os.rename(part_path, dst_path)
        except BaseException:
            try:
                os.remove(part_path)
            except Exception:
                pass
            raise
```

`stream=True` tells requests *not* to load the whole response body in memory (some podcasts can be a few hundred MB in size); instead, we copy the content from the underlying file-like object to disk using `shutil.copyfileobj()`.

In order to avoid leaving around incomplete files in case of failure, we first write the content to a temporary file which we try to delete if anything goes wrong. After we finish writing the content successfully, we move the temporary file to its final destination.

### 3.3.5 Wrapping up

We're mostly done.

Uncomment the `add_and_update_feed()` call, remove the `entries = list(entries)[:3]` line in download_everything(), and clean up the files we created so we can start over for real:

```
rm -r db.sqlite podcasts/
```

The script output should now look like:

```
updated Hello Internet (last changed at 2020-02-28 09:34:02+00:00)

Hello Internet - H.I. #136: Dog Bingo
  * 136FinalFinal.mp3
Hello Internet - H.I. #135: Place Your Bets
  * 135.mp3
Hello Internet - # H.I. 134: Boxing Day
  * HI134.mp3
...
```

with actual MP3 files being downloaded to *podcasts/* (which takes a while).

If you interrupt the script at any point (CTRL+C), it should start from the first episode it did not download. If you let it finish and run it again, it will only update the feed (unless a new episode just came up; then it will download it).

### 3.3.6 More examples

You can find more examples of how to use *reader* in the repository:

- download all new episodes of a podcast (the script from this tutorial)
- a simple terminal feed reader

## 3.4 User guide

This page gives a tour of *reader*'s features, and a few examples of how to use them.

**Note:** Before starting, make sure that *reader* is *installed* and up-to-date.

### 3.4.1 The Reader object

Most *reader* functionality is available through a *Reader* instance, which persists feed and entry state and provides operations on them; in MVC (model–view–controller) parlance, you would probably call it a fat model.

To create a new Reader, call *make_reader()* with the path to a database file:

```
>>> from reader import make_reader
>>> reader = make_reader("db.sqlite")
```

The default (and currently only) storage uses SQLite, so the path behaves like the `database` argument of `sqlite3.connect()`:

- If the database does not exist, it will be created automatically.
- You can pass `":memory:"` to use a temporary in-memory database; the data will disappear when the reader is closed.

#### Lifecycle

In order to perform maintenance tasks and release underlying resources in a predictable manner, you should use the reader as a context manager:

```
with make_reader('db.sqlite') as reader:
    ... # do stuff with reader
```

For convenience, you can also use the reader directly. In this case, maintenance tasks may sometimes (rarely) be performed before arbitrary method calls return. You can still release the underlying resources by calling *close()*. `with reader` is roughly equivalent to `with contextlib.closing(reader)`, but the former suspends regular maintenance tasks for the duration of the with block.

In either case, you can reuse the reader object after closing it; database connections will be re-created automatically.

### Threading

You can use the same reader instance from multiple threads:

```
>>> Thread(target=reader.update_feeds).start()
```

You should use the reader as a context manager or call its *close()* method *from each thread* where it is used.

It is not always possible to close the reader from your code, especially when you do not control how threads are shut down – for example, if you want to use a reader across requests in a Flask web application, or with a ThreadPoolExecutor. If you do not close the reader, it will attempt to call *close()* before the thread ends. Currently, this does not work on PyPy, or if the thread was not created through the threading module (but note that database connections will eventually be closed anyway when garbage-collected).

### Temporary databases

To maximize the usefulness of temporary databases, the database connection is closed (and the data discarded) only when calling *close()*, not when using the reader as a context manager. The reader cannot be reused after calling *close()*.

```
>>> reader = make_reader(':memory:')
>>> with reader:
...     reader.set_tag((), 'tag')
...
>>> list(reader.get_tag_keys(()))
['tag']
>>> reader.close()
>>> list(reader.get_tag_keys(()))
Traceback (most recent call last):
  ...
reader.exceptions.StorageError: usage error: cannot reuse a private database after␣
↪close()
```

It is not possible to use a private, temporary SQLite database from other threads, since each connection would be to a *different* database:

```
>>> Thread(target=reader.update_feeds).start()
Exception in thread Thread-1 (update_feeds):
Traceback (most recent call last):
  ...
reader.exceptions.StorageError: usage error: cannot use a private database from threads␣
↪other than the creating thread
```

### Back-ups

Making back-ups depends on the storage used.

For the SQLite storage, you should use the sqlite3 .backup command or VACUUM INTO (see backup.sh for an example).

### 3.4.2 Adding feeds

To add a feed, call the *add_feed()* method with the feed URL:

```
>>> reader.add_feed("https://www.relay.fm/cortex/feed")
>>> reader.add_feed("http://www.hellointernet.fm/podcast?format=rss")
```

Most of the attributes of a new feed are empty (to populate them, the feed must be *updated*):

```
>>> feed = reader.get_feed("http://www.hellointernet.fm/podcast?format=rss")
>>> print(feed)
Feed(url='http://www.hellointernet.fm/podcast?format=rss', updated=None, title=None, ...)
```

### 3.4.3 File-system access

*reader* supports *http(s)://* and local (*file:*) feeds.

For security reasons, local feeds are disabled by default. You can allow full file-system access or restrict it to a single directory by using the `feed_root` *make_reader()* argument:

```
>>> # all local feed paths allowed
>>> reader = make_reader("db.sqlite", feed_root='')
>>> # local feed paths are relative to /feeds
>>> reader = make_reader("db.sqlite", feed_root='/feeds')
>>> # ok, resolves to /feeds/feed.xml
>>> reader.add_feed("feed.xml")
>>> # ok, resolves to /feeds/also/feed.xml
>>> reader.add_feed("file:also/feed.xml")
>>> # error, resolves to /feed.xml, which is above /feeds
>>> reader.add_feed("file:../feed.xml")
Traceback (most recent call last):
  ...
ValueError: path cannot be outside root: '/feed.xml'
```

Note that it is possible to add invalid feeds; *updating* them will still fail, though:

```
>>> reader.add_feed("file:../feed.xml", allow_invalid_url=True)
>>> reader.update_feed("file:../feed.xml")
Traceback (most recent call last):
  ...
reader.exceptions.ParseError: path cannot be outside root: '/feed.xml': 'file:../feed.xml
↪'
```

### 3.4.4 Deleting feeds

To delete a feed and all the data associated with it, use *delete_feed()*:

```
>>> reader.delete_feed("https://www.example.com/feed.xml")
```

### 3.4.5 Updating feeds

To retrieve the latest version of a feed, along with any new entries, it must be updated. You can update all the feeds by using the *update_feeds()* method:

```
>>> reader.update_feeds()
>>> reader.get_feed(feed)
Feed(url='http://www.hellointernet.fm/podcast?format=rss', updated=datetime.
↪datetime(2020, 2, 28, 9, 34, 2, tzinfo=datetime.timezone.utc), title='Hello Internet',␣
↪...)
```

To retrive feeds in parallel, use the `workers` flag:

```
>>> reader.update_feeds(workers=10)
```

You can also update a specific feed using *update_feed()*:

```
>>> reader.update_feed("http://www.hellointernet.fm/podcast?format=rss")
```

If supported by the server, *reader* uses the ETag and Last-Modified headers to only retrieve feeds if they changed (details). Even so, you should not update feeds *too* often, to avoid wasting the feed publisher's resources, and potentially getting banned; every 30 minutes seems reasonable.

To support updating newly-added feeds off the regular update schedule, you can use the `new_only` flag; you can call this more often (e.g. every minute):

```
>>> reader.update_feeds(new_only=True)
```

If you need the status of each feed as it gets updated (for instance, to update a progress bar), you can use *update_feeds_iter()* instead, and get a (url, updated feed or none or exception) pair for each feed:

```
>>> for url, value in reader.update_feeds_iter():
...     if value is None:
...         print(url, "not modified")
...     elif isinstance(value, Exception):
...         print(url, "error:", value)
...     else:
...         print(url, value.new, "new,", value.updated, "updated")
...
http://www.hellointernet.fm/podcast?format=rss 100 new, 0 updated
https://www.relay.fm/cortex/feed not modified
```

### 3.4.6 Disabling feed updates

Sometimes, it is useful to skip a feed when using *update_feeds()*; for example, the feed does not exist anymore, and you want to stop requesting it unnecessarily during regular updates, but still want to keep its entries (so you cannot remove it).

*disable_feed_updates()* allows you to do exactly that:

```
>>> reader.disable_feed_updates(feed)
```

You can check if updates are enabled for a feed by looking at its *updates_enabled* attribute:

```
>>> reader.get_feed(feed).updates_enabled
False
```

### 3.4.7 Getting feeds

As seen in the previous sections, *get_feed()* returns a *Feed* object with more information about a feed:

```
>>> from prettyprinter import pprint, install_extras;
>>> install_extras(include=['dataclasses'])
>>> feed = reader.get_feed(feed)
>>> pprint(feed)
reader.types.Feed(
    url='http://www.hellointernet.fm/podcast?format=rss',
    updated=datetime.datetime(
        year=2020,
        month=2,
        day=28,
        hour=9,
        minute=34,
        second=2,
        tzinfo=datetime.timezone.utc
    ),
    title='Hello Internet',
    link='http://www.hellointernet.fm/',
    author='CGP Grey',
    added=datetime.datetime(2020, 10, 12, tzinfo=datetime.timezone.utc),
    last_updated=datetime.datetime(2020, 10, 12, tzinfo=datetime.timezone.utc)
)
```

To get all the feeds, use the *get_feeds()* method:

```
>>> for feed in reader.get_feeds():
...     print(
...         feed.title or feed.url,
...         f"by {feed.author or 'unknown author'},",
...         f"updated on {feed.updated or 'never'}",
...     )
...
Cortex by Relay FM, updated on 2020-09-14 12:15:00+00:00
Hello Internet by CGP Grey, updated on 2020-02-28 09:34:02+00:00
```

*get_feeds()* also allows filtering feeds by their *tags*, if the last update succeeded, or if updates are enabled, and changing the feed sort order.

### 3.4.8 Changing feed URLs

Sometimes, feeds move from one URL to another.

This can be handled naively by removing the old feed and adding the new URL; however, all the data associated with the old feed would get lost, including any old entries (some feeds only have the last X entries).

To change the URL of a feed in-place, use *change_feed_url()*:

```
>>> reader.change_feed_url(
...     "https://www.example.com/old.xml",
...     "https://www.example.com/new.xml"
... )
```

Sometimes, the id of the entries changes as well; you can handle duplicates by using the *entry_dedupe* plugin.

### 3.4.9 Getting entries

You can get all the entries, most-recent first, by using *get_entries()*, which generates *Entry* objects:

```
>>> for entry, _ in zip(reader.get_entries(), range(10)):
...     print(entry.feed.title, '-', entry.title)
...
Cortex - 106: Clear and Boring
...
Hello Internet - H.I. #136: Dog Bingo
```

*get_entries()* allows filtering entries by their feed, *flags*, *feed tags*, or enclosures, and changing the entry sort order. Here is an example of getting entries for a single feed:

```
>>> feed.title
'Hello Internet'
>>> entries = list(reader.get_entries(feed=feed))
>>> for entry in entries[:2]:
...     print(entry.feed.title, '-', entry.title)
...
Hello Internet - H.I. #136: Dog Bingo
Hello Internet - H.I. #135: Place Your Bets
```

### 3.4.10 Entry flags

Entries can be marked as *read* or *important*. These flags can be used for filtering:

```
>>> reader.mark_entry_as_read(entries[0])
>>> entries = list(reader.get_entries(feed=feed, read=False))
>>> for entry in entries[:2]:
...     print(entry.title)
...
H.I. #135: Place Your Bets
# H.I. 134: Boxing Day
```

The time when a flag was changed is available via *read_modified* and *important_modified*:

```
>>> for entry in reader.get_entries(feed=feed, limit=2):
...     print(entry.title, '-', entry.read, entry.read_modified)
...
H.I. #136: Dog Bingo - True 2021-10-08 08:00:00+00:00
H.I. #135: Place Your Bets - False None
```

### 3.4.11 Full-text search

*reader* supports full-text searches over the entries' content through the *search_entries()* method.

```
>>> reader.update_search()
>>> for result in reader.search_entries('mars'):
...     print(result.metadata['.title'].apply('*', '*'))
...
H.I. #106: Water on *Mars*
```

*search_entries()* generates *EntrySearchResult* objects containing snippets of relevant entry/feed fields, with the parts that matched highlighted.

By default, results are filtered by relevance; you can sort them most-recent first by passing `sort='recent'`. Also, you can filter them just as with *get_entries()*.

The search index is not updated automatically; to keep it in sync, you need to call *update_search()* when entries change (e.g. after updating/deleting feeds). *update_search()* only updates the entries that changed since the last call, so it is OK to call it relatively often.

Search can be turned on/off through the *enable_search()* / *disable_search()* methods (persistent across instances using the same database), or the `search_enabled` argument of *make_reader()*; by default, search is enabled automatically on the first *update_search()* call. If search is enabled, you should call *update_search()* regularly to prevent unprocesses changes from accumulating over time.

Because the search index can be almost as large as the main database, the default implementation splits it into a separate, attached database, which allows *backing up* the main database separately; for a reader created with `make_reader('db.sqlite')`, the search index will be in `db.sqlite.search`.

Changed in version 3.12: Split the full-text search index into a separate database.

### 3.4.12 Resource tags

Resources (feeds and entries) can have tags, key-value pairs where the values are any JSON-serializable data:

```
>>> reader.get_tag(feed, 'one', 'default')
'default'
>>> reader.set_tag(feed, 'one', 'value')
>>> reader.get_tag(feed, 'one')
'value'
>>> reader.set_tag(feed, 'two', {2: ['ii']})
>>> dict(reader.get_tags(feed))
{'one': 'value', 'two': {'2': ['ii']}}
```

Common uses for tag values are plugin and UI settings.

In addition to feeds and entries, it is possible to store global (per-database) data. To work with global tags, use `()` (the empty tuple) as the first argument of the tag methods.

When using *set_tag()*, the value can be omitted, in which case the behavior is to ensure the tag exists (if it doesn't, None is used as value):

```
>>> reader.set_tag(feed, 'two')
>>> reader.set_tag(feed, 'three')
>>> set(reader.get_tag_keys(feed))
{'three', 'one', 'two'}
>>> dict(reader.get_tags(feed))
{'one': 'value', 'three': None, 'two': {'2': ['ii']}}
```

Besides storing resource metadata, tags can be used for filtering feeds and entries (see *TagFilterInput* for more complex examples):

```
>>> # feeds that have the tag "one"
>>> [f.title for f in reader.get_feeds(tags=['one'])]
['Hello Internet']
>>> # entries of feeds that have no tags
>>> [
...     (e.feed.title, e.title)
...     for e in reader.get_entries(feed_tags=[False])
... ][:2]
[('Cortex', '106: Clear and Boring'), ('Cortex', '105: Atomic Notes')]
```

Note that tag keys and the top-level keys of dict tag values starting with specific (configurable) prefixes are *reserved*. Other than that, they can be any unicode string, although UIs might want to restrict this to a smaller set of characters.

Changed in version 2.10: Support entry and global tags.

Changed in version 2.8: Prior to version 2.7, there were two separate APIs, with independent namespaces:

- feed metadata (key/value pairs, could *not* be used for filtering)
- feed tags (plain strings, could be used for filtering)

In version 2.7, the two namespaces were merged (such that adding a tag to a feed would result in the metadata with the same key being set with a value of None).

In version 2.8, these separate APIs were merged into a new, unified API for generic resource tags (key/value pairs which can be used for filtering). The old, feed-only tags/metadata methods were deprecated, and **will be removed in version 3.0**.

### 3.4.13 Counting things

You can get aggregated feed and entry counts by using one of the *get_feed_counts()*, *get_entry_counts()*, or *search_entry_counts()* methods:

```
>>> reader.get_feed_counts()
FeedCounts(total=156, broken=5, updates_enabled=154)
>>> reader.get_entry_counts()
EntryCounts(total=12494, read=10127, important=115, has_enclosures=2823, averages=...)
>>> reader.search_entry_counts('feed: death and gravity')
EntrySearchCounts(total=16, read=16, important=0, has_enclosures=0, averages=...)
```

The _counts methods support the same filtering arguments as their non-_counts counterparts. The following example shows how to get counts only for feeds/entries with a specific tag:

```
>>> for tag in itertools.chain(reader.get_tag_keys((None,)), [False]):
...     feeds = reader.get_feed_counts(tags=[tag])
...     entries = reader.get_entry_counts(feed_tags=[tag])
...     print(f"{tag or '<no tag>'}: {feeds.total} feeds, {entries.total} entries ")
...
podcast: 27 feeds, 2838 entries
python: 39 feeds, 1929 entries
self: 5 feeds, 240 entries
tech: 90 feeds, 7075 entries
webcomic: 6 feeds, 1865 entries
<no tag>: 23 feeds, 1281 entries
```

For entry counts, the *averages* attribute is the average number of entries per day during the last 1, 3, 12 months, as a 3-tuple (e.g. to get an idea of how often a feed gets updated):

```
>>> reader.get_entry_counts().averages
(8.066666666666666, 8.054945054945055, 8.446575342465753)
>>> reader.search_entry_counts('feed: death and gravity').averages
(0.03333333333333333, 0.06593406593406594, 0.043835616438356165)
```

This example shows how to convert them to monthly statistics:

```
>>> periods = [(30, 1, 'month'), (91, 3, '3 months'), (365, 12, 'year')]
>>> for avg, (days, months, label) in zip(counts.averages, periods):
...     entries = round(avg * days / months, 1)
...     print(f"{entries} entries/month (past {label})")
...
1.0 entries/month (past month)
2.0 entries/month (past 3 months)
1.3 entries/month (past year)
```

### 3.4.14 Deleting entries

As of version 3.12, entries are **not** deleted automatically, and there is no high-level way of deleting entries; see #96 for details and updates.

Deleting entries properly is non-trivial for two reasons:

- Deleted entries should stay deleted; right now, if you delete an entry that still appears in the feed, it will be added again on the next update.
- The *entry_dedupe* plugin needs the old entry in order to work.

If you do not care about these issues, you can delete entries using the low-level `delete_entries()` storage method.

### 3.4.15 Pagination

*get_feeds()*, *get_entries()*, and *search_entries()* can be used in a paginated fashion.

The `limit` argument allows limiting the number of results returned; the `starting_after` argument allows skipping results until after a specific one.

To get the first page, use only `limit`:

```
>>> for entry in reader.get_entries(limit=2):
...     print(entry.title)
...
H.I. #136: Dog Bingo
H.I. #135: Place Your Bets
```

To get the next page, use the last result from a call as `starting_after` in the next call:

```
>>> for entry in reader.get_entries(limit=2, starting_after=entry):
...     print(entry.title)
...
# H.I. 134: Boxing Day
Star Wars: The Rise of Skywalker, Hello Internet Christmas Special
```

### 3.4.16 Plugins

*reader* supports plugins as a way to extend its default behavior.

To use a built-in plugin, pass the plugin name to *make_reader()*:

```
>>> reader = make_reader("db.sqlite", plugins=[
...     "reader.enclosure_dedupe",
...     "reader.entry_dedupe",
... ])
```

You can find the full list of built-in plugins *here*, and the list of plugins used by default in *reader.plugins.DEFAULT_PLUGINS*.

#### Custom plugins

In addition to built-in plugins, reader also supports *custom plugins*.

A custom plugin is any callable that takes a *Reader* instance and potentially modifies it in some (useful) way. To use custom plugins, pass them to *make_reader()*:

```
>>> def function_plugin(reader):
...     print(f"got {reader}")
...
>>> class ClassPlugin:
...     def __init__(self, **options):
...         self.options = options
...     def __call__(self, reader):
...         print(f"got options {self.options} and {reader}")
...
>>> reader = make_reader("db.sqlite", plugins=[
```

(continues on next page)

(continued from previous page)

```
...     function_plugin,
...     ClassPlugin(option=1),
... ])
got <reader.core.Reader object at 0x7f8897824a00>
got options {'option': 1} and <reader.core.Reader object at 0x7f8897824a00>
```

For a real-world example, see the implementation of the enclosure_dedupe built-in plugin. Using it as a custom plugin looks like this:

```
>>> from reader.plugins import enclosure_dedupe
>>> reader = make_reader("db.sqlite", plugins=[enclosure_dedupe.init_reader])
```

### 3.4.17 Feed and entry arguments

As you may have noticed in the examples above, feed URLs and *Feed* objects can be used interchangeably as method arguments. This is by design. Likewise, wherever an entry argument is expected, you can either pass a *(feed URL, entry id)* tuple or an *Entry* (or *EntrySearchResult*) object.

You can get this unique identifier in a uniform way by using the *resource_id* property. This is useful when you need to refer to a *reader* object in a generic way from outside Python (e.g. to make a link to the next *page* of feeds/entries in a web application).

### 3.4.18 Streaming methods

All methods that return iterators (*get_feeds()*, *get_entries()* etc.) generate the results lazily.

Some examples of how this is useful:

- Consuming the first 100 entries should take *roughly* the same amount of time, whether you have 1000 or 100000 entries.
- Likewise, if you don't keep the entries around (e.g. append them to a list), memory usage should remain relatively constant regardless of the total number of entries returned.

### 3.4.19 Reserved names

In order to expose *reader* and plugin functionality directly to the end user, *names* starting with `.reader.` and `.plugin.` are *reserved*. This applies to the following names:

- tag keys
- the top-level keys of dict tag values

Currently, there are no *reader*-reserved names; new ones will be documented here.

The prefixes can be changed using *reserved_name_scheme*.

Note that changing *reserved_name_scheme* *does not rename* the actual entities, it just controls how new reserved names are built. Because of this, I recommend choosing a scheme before setting up a new *reader* database, and sticking with that scheme for its lifetime. To change the scheme of an existing database, you must rename the entities listed above yourself.

When choosing a *reserved_name_scheme*, the `reader_prefix` and `plugin_prefix` should not overlap, otherwise the *reader* core and various plugins may interfere each other. (For example, if both prefixes are set to `.`, *reader*-reserved

key `user_title` and a plugin named `user_title` that uses just the plugin name (with no key) will both end up using the `.user_title` tag.)

That said, *reader* will ensure names reserved by the core and *built-in plugin* names *will never collide*, so this is a concern only if you plan to use third-party plugins.

Reserved names can be built programmatically using *make_reader_reserved_name()* and *make_plugin_reserved_name()*. Code that wishes to work with any scheme should always use these methods to construct reserved names (especially third-party plugins).

### 3.4.20 Advanced feedparser features

*reader* uses feedparser ("Universal Feed Parser") to parse feeds. It comes with a number of advanced features, most of which *reader* uses transparently.

Two of these features are worth mentioning separately, since they change the content of the feed, and, although *always enabled* at the moment, they may become optional in the future; note that disabling them is not currently possible.

#### Sanitization

Quoting:

> Most feeds embed HTML markup within feed elements. Some feeds even embed other types of markup, such as SVG or MathML. Since many feed aggregators use a web browser (or browser component) to display content, Universal Feed Parser sanitizes embedded markup to remove things that could pose security risks.

You can find more details about which markup and elements are sanitized in the feedparser documentation.

The following corresponding *reader* attributes are sanitized:

- *Entry.content* (*Content.value*)
- *Entry.summary*
- *Entry.title*
- *Feed.title*

#### Relative link resolution

Quoting:

> Many feed elements and attributes are URIs. Universal Feed Parser resolves relative URIs according to the XML:Base specification. [...]
>
> In addition [to elements treated as URIs], several feed elements may contain HTML or XHTML markup. Certain elements and attributes in HTML can be relative URIs, and Universal Feed Parser will resolve these URIs according to the same rules as the feed elements listed above.

You can find more details about which elements are treated as URIs and HTML markup in the feedparser documentation.

The following corresponding *reader* attributes are treated as URIs:

- *Entry.enclosures* (*Enclosure.href*)
- *Entry.id*
- *Entry.link*

- *Feed.link*

The following corresponding *reader* attributes may be treated as HTML markup, depending on their type attribute or feedparser defaults:

- *Entry.content* (*Content.value*)
- *Entry.summary*
- *Entry.title*
- *Feed.title*

### 3.4.21 Errors and exceptions

All exceptions that *Reader* explicitly raises inherit from *ReaderError*.

If there's an issue retrieving or parsing the feed, *update_feed()* will raise a *ParseError* with the original exception (if any) as cause. *update_feeds()* will just log the exception and move on. In both cases, information about the cause will be stored on the feed in *last_exception*.

Any unexpected exception raised by the underlying storage implementation will be reraised as a *StorageError*, with the original exception as cause.

Search methods will raise a *SearchError*. Any unexpected exception raised by the underlying search implementation will be also be reraised as a *SearchError*, with the original exception as cause.

When trying to create a feed, entry, or tag that already exists, or to operate on one that does not exist, a corresponding `*ExistsError` or `*NotFoundError` will be raised.

All functions and methods may raise `ValueError` or `TypeError` implicitly or explicitly if passed invalid arguments.

CHAPTER

FOUR

# API REFERENCE

If you are looking for information on a specific function, class, or method, this part of the documentation is for you.

## 4.1 API reference

This part of the documentation covers all the public interfaces of *reader*.

### 4.1.1 Reader object

Most of *reader*'s functionality can be accessed through a `Reader` instance.

`reader.`**`make_reader`**(*url*, ***, *feed_root=None*, *plugins=...*, *session_timeout=(3.05, 60)*, *reserved_name_scheme=...*, *search_enabled='auto'*)

Create a new `Reader`.

*reader* can optionally parse local files, with the feed URL either a bare path or a file URI.

The interpretation of local feed URLs depends on the value of the feed `feed_root` argument. It can be one of the following:

`None`

No local file parsing. Updating local feeds will fail.

`''` (the empty string)

Full filesystem access. This should be used only if the source of feed URLs is trusted.

Both absolute and relative feed paths are supported. The current working directory is used normally (as if the path was passed to `open()`).

Example: Assuming the current working directory is `/feeds`, all of the following feed URLs correspond to `/feeds/feed.xml`: `feed.xml`, `/feeds/feed.xml`, `file:feed.xml`, and `file:/feeds/feed.xml`.

`'/path/to/feed/root'` (any non-empty string)

An absolute path; all feed URLs are interpreted as relative to it. This can be used if the source of feed URLs is untrusted.

Feed paths must be relative. The current working directory is ignored.

Example: Assuming the feed root is `/feeds`, feed URLs `feed.xml` and `file:feed.xml` correspond to `/feeds/feed.xml`. `/feed.xml` and `file:/feed.xml` are both errors.

Relative paths pointing outside the feed root are errors, to prevent directory traversal attacks. Note that symbolic links inside the feed root *can* point outside it.

The root and feed paths are joined and normalized with no regard for symbolic links; see `os.path.normpath()` for details.

Accessing device files on Windows is an error.

**Parameters**

- **url** (*str*) – Path to the reader database.
- **feed_root** (*str or None*) – Directory where to look for local feeds. One of `None` (don't open local feeds; default), `''` (full filesystem access), or `'/path/to/feed/root'` (an absolute path that feed paths are relative to).
- **plugins** (*iterable(str or callable(Reader)) or None*) – An iterable of built-in plugin names or *plugin(reader) –> None* callables. The callables are called with the reader object before it is returned. Exceptions from plugin code will propagate to the caller. Defaults to *DEFAULT_PLUGINS*.
- **session_timeout** (*float or tuple(float, float) or None*) – When retrieving HTTP(S) feeds, how many seconds to wait for the server to send data, as a float, or a (connect timeout, read timeout) tuple. Passed to the underlying Requests session.
- **reserved_name_scheme** (*dict(str, str)*) – Value for *reserved_name_scheme*. The prefixes default to `.reader./.plugin.`, and the separator to `.`
- **search_enabled** (bool or None or `'auto'`) – Whether to enable search. One of `'auto'` (enable on the first *update_search()* call; default), `True` (enable), `False` (disable), `None` (do nothing).

**Returns**
The reader.

**Return type**
*Reader*

**Raises**

- ***StorageError*** – An error occurred while connecting to storage.
- ***SearchError*** – An error occurred while enabling/disabling search.
- ***InvalidPluginError*** – An invalid plugin name was passed to `plugins`.
- ***PluginInitError*** – A plugin failed to initialize.
- ***PluginError*** – An ambiguous plugin-related error occurred.
- ***ReaderError*** – An ambiguous exception occurred while creating the reader.

Changed in version 3.0: Wrap exceptions raised during plugin initialization in *PluginInitError* instead of letting them bubble up.

New in version 2.4: The `search_enabled` keyword argument.

Changed in version 2.4: Enable search on the first *update_search()* call. To get the previous behavior (leave search as-is), use `search_enabled=None`.

Changed in version 2.0: `feed_root` now defaults to `None` (don't open local feeds) instead of `''` (full filesystem access).

New in version 1.17: The `reserved_name_scheme` keyword argument.

New in version 1.16: The `plugins` keyword argument. Using an invalid plugin name raises *InvalidPluginError*, a `ValueError` subclass.

New in version 1.14: The `session_timeout` keyword argument, with a default of (3.05, 60) seconds; the previous behavior was to *never time out*.

New in version 1.6: The `feed_root` keyword argument.

**class** reader.**Reader**(...)

A feed reader.

Persists feed and entry state, provides operations on them, and stores configuration.

Currently, the following feed types are supported:

- Atom (provided by feedparser)
- RSS (provided by feedparser)
- JSON Feed

Additional sources can be added through *plugins*.

In order to perform maintenance tasks and release underlying resources in a predictable manner, the Reader object should be used as a context manager *from each thread* where it is used. For convenience, it is possible to use a Reader object directly; in this case, maintenance tasks may sometimes be performed before arbitrary method calls return.

---

**Important:** Reader objects should be created using *make_reader()*; the Reader constructor is not stable yet and may change without any notice.

---

Changed in version 2.16: Allow using a Reader object from multiple threads directly (do not require it to be used as a context manager anymore).

Changed in version 2.16: Allow Reader objects to be reused after closing.

Changed in version 2.16: Allow using a Reader object from multiple asyncio tasks.

Changed in version 2.15: Allow using Reader objects as context managers.

Changed in version 2.15: Allow using Reader objects from threads other than the creating thread.

Changed in version 2.10: Allow passing a *(feed URL,)* 1-tuple anywhere a feed URL can be passed.

New in version 1.13: JSON Feed support.

**close**()

Close this *Reader*.

Releases any underlying resources associated with the reader.

The reader can be reused after being closed (but you have to call close() again after that).

close() should be called *from each thread* where the reader is used. Prefer using the reader as a context manager instead.

**Raises**
*ReaderError* –

Changed in version 2.16: Allow calling close() from any thread.

**add_feed**(*feed*, */*, *exist_ok=False*, ***, *allow_invalid_url=False*)

Add a new feed.

Feed updates are enabled by default.

**Parameters**

- **feed** (*str or tuple(str) or Feed*) – The feed URL.
- **allow_invalid_url** (*bool*) – Add feed even if the current Reader configuration does not know how to handle the feed URL (and updates for it would fail).
- **exist_ok** (*bool*) – If true, don't raise *FeedExistsError* if the feed already exists.

**Raises**

- ***FeedExistsError*** – If the feed already exists, and *exist_ok* is false.
- ***StorageError*** –
- ***InvalidFeedURLError*** – If `feed` is invalid and `allow_invalid_url` is false.

Changed in version 3.0: The `feed` argument is now positional-only.

New in version 2.8: The `exist_ok` argument.

New in version 2.5: The `allow_invalid_url` keyword argument.

Changed in version 2.5: Validate the new feed URL. To get the previous behavior (no validation), use `allow_invalid_url=True`.

**delete_feed**(*feed*, */*, *missing_ok=False*)

Delete a feed and all of its entries and tags.

**Parameters**

- **feed** (*str or tuple(str) or Feed*) – The feed URL.
- **missing_ok** (*bool*) – If true, don't raise *FeedNotFoundError* if the feed does not exist.

**Raises**

- ***FeedNotFoundError*** – If the feed does not exist, and *missing_ok* is false.
- ***StorageError*** –

Changed in version 3.0: The `feed` argument is now positional-only.

New in version 2.8: The `missing_ok` argument.

New in version 1.18: Renamed from `remove_feed()`.

**change_feed_url**(*old*, *new*, */*, ***, *allow_invalid_url=False*)

Change the URL of a feed.

User-defined feed attributes are preserved: *added*, *user_title*. Feed-defined feed attributes are also preserved, at least until the next update: *title*, *link*, *author*, *subtitle* (except *updated* and *version*, which get set to None). All other feed attributes are set to their default values.

The entries and tags are preserved.

**Parameters**

- **old** (*str or tuple(str) or Feed*) – The old feed; must exist.
- **new** (*str or tuple(str) or Feed*) – The new feed; must not exist.

- **allow_invalid_url** (*bool*) – Change feed URL even if the current Reader configuration does not know how to handle the new feed URL (and updates for it would fail).

**Raises**

- *FeedNotFoundError* – If `old` does not exist.
- *FeedExistsError* – If `new` already exists.
- *StorageError* –
- *InvalidFeedURLError* – If `new` is invalid and `allow_invalid_url` is false.

Changed in version 3.0: The `old` and `new` arguments are now positional-only.

New in version 2.5: The `allow_invalid_url` keyword argument.

Changed in version 2.5: Validate the new feed URL. To get the previous behavior (no validation), use `allow_invalid_url=True`.

New in version 1.8.

**get_feeds**(*, *feed=None*, *tags=None*, *broken=None*, *updates_enabled=None*, *new=None*, *sort='title'*, *limit=None*, *starting_after=None*)

Get all or some of the feeds.

**Parameters**

- **feed** (*str or tuple(str) or Feed or None*) – Only return the feed with this URL.
- **tags** (*None or bool or list(str or bool or list(str or bool))*) – Only return feeds matching these tags; see *TagFilterInput* for details.
- **broken** (*bool or None*) – Only return broken / healthy feeds.
- **updates_enabled** (*bool or None*) – Only return feeds that have updates enabled / disabled.
- **new** (*bool or None*) – Only return feeds that have never been updated / have been updated before.
- **sort** (*str*) – How to order feeds; one of `'title'` (by *user_title* or *title*, case insensitive; default), or `'added'` (last added first).
- **limit** (*int or None*) – A limit on the number of feeds to be returned; by default, all feeds are returned.
- **starting_after** (*str or tuple(str) or Feed or None*) – Return feeds after this feed; a cursor for use in pagination.

**Yields**

*Feed* – Sorted according to `sort`.

**Raises**

- *StorageError* –
- *FeedNotFoundError* – If `starting_after` does not exist.

New in version 2.6: The `new` keyword argument.

New in version 1.12: The `limit` and `starting_after` keyword arguments.

New in version 1.11: The `updates_enabled` keyword argument.

New in version 1.7: The `tags` keyword argument.

New in version 1.7: The `broken` keyword argument.

**get_feed**(*feed: str | FeedLike*, */*) → *Feed*

**get_feed**(*feed: str | FeedLike*, *default: _T*, */*) → *Feed* | _T

Get a feed.

Like `next(iter(reader.get_feeds(feed=feed)))`, but raises a custom exception instead of `StopIteration`.

**Parameters**

- **feed** (*str or tuple(str) or Feed*) – The feed URL.
- **default** – Returned if given and the feed does not exist.

**Returns**
The feed.

**Return type**
*Feed*

**Raises**

- ***FeedNotFoundError*** –
- ***StorageError*** –

Changed in version 3.0: The `feed` and `default` arguments are now positional-only.

**get_feed_counts**(***, *feed=None*, *tags=None*, *broken=None*, *updates_enabled=None*, *new=None*)

Count all or some of the feeds.

**Parameters**

- **feed** (*str or tuple(str) or Feed or None*) – Only count the feed with this URL.
- **tags** (*None or bool or list(str or bool or list(str or bool))*) – Only count feeds matching these tags; see *TagFilterInput* for details.
- **broken** (*bool or None*) – Only count broken / healthy feeds.
- **updates_enabled** (*bool or None*) – Only count feeds that have updates enabled / disabled.
- **new** (*bool or None*) – Only count feeds that have never been updated / have been updated before.

**Return type**
*FeedCounts*

**Raises**
***StorageError*** –

New in version 2.6: The `new` keyword argument.

New in version 1.11.

**set_feed_user_title**(*feed*, *title*, */*)

Set a user-defined title for a feed.

**Parameters**

- **feed** (*str or tuple(str) or Feed*) – The feed URL.
- **title** (*str or None*) – The title, or None to remove the current title.

**Raises**

- *FeedNotFoundError* –
- *StorageError* –

Changed in version 3.0: The `feed` and `title` arguments are now positional-only.

**enable_feed_updates**(*feed*, */*)

Enable updates for a feed.

See *update_feeds()* for details.

**Parameters**
**feed** (*str or tuple(str) or Feed*) – The feed URL.

**Raises**

- *FeedNotFoundError* –
- *StorageError* –

Changed in version 3.0: The `feed` argument is now positional-only.

New in version 1.11.

**disable_feed_updates**(*feed*, */*)

Disable updates for a feed.

See *update_feeds()* for details.

**Parameters**
**feed** (*str or tuple(str) or Feed*) – The feed URL.

**Raises**

- *FeedNotFoundError* –
- *StorageError* –

Changed in version 3.0: The `feed` argument is now positional-only.

New in version 1.11.

**update_feeds**(**, feed=None, tags=None, broken=None, updates_enabled=True, new=None, workers=1*)

Update all or some of the feeds.

Silently skip feeds that raise *ParseError*.

Re-raise *before_feeds_update_hooks* failures immediately. Collect all other update hook failures and re-raise them as an *UpdateHookErrorGroup*; currently, only the exceptions for the first 5 feeds with hook failures are collected.

By default, update all the feeds that have updates enabled.

Roughly equivalent to `for _ in reader.update_feeds_iter(...): pass`.

**Parameters**

- **feed** (*str or tuple(str) or Feed or None*) – Only update the feed with this URL.
- **tags** (*None or bool or list(str or bool or list(str or bool))*) – Only update feeds matching these tags; see *TagFilterInput* for details.
- **broken** (*bool or None*) – Only update broken / healthy feeds.
- **updates_enabled** (*bool or None*) – Only update feeds that have updates enabled / disabled. Defaults to true.

- **new** (*bool or None*) – Only update feeds that have never been updated / have been updated before. Defaults to None.
- **workers** (*int*) – Number of threads to use when getting the feeds.

**Raises**

- ***UpdateHookError*** – For unexpected hook exceptions.
- ***UpdateError*** –
- ***StorageError*** –

Changed in version 3.8: Wrap unexpected update hook exceptions in *UpdateHookError*. Try to update all the feeds, don't stop after a feed/entry hook fails.

Changed in version 3.8: Document this method can raise non-feed-related *UpdateError*s (other than *UpdateHookError*).

New in version 2.6: The `feed`, `tags`, `broken`, and `updates_enabled` keyword arguments.

Changed in version 2.0: Removed the `new_only` parameter.

Changed in version 2.0: All parameters are keyword-only.

Changed in version 1.15: Update entries whenever their content changes, regardless of their *updated* date.

Content-only updates (not due to an *updated* change) are limited to 24 consecutive updates, to prevent spurious updates for entries whose content changes excessively (for example, because it includes the current time).

Previously, entries would be updated only if the entry *updated* was *newer* than the stored one.

Changed in version 1.11: Only update the feeds that have updates enabled.

**update_feeds_iter**(**, feed=None, tags=None, broken=None, updates_enabled=True, new=None, workers=1, _call_feeds_update_hooks=True*)

Update all or some of the feeds.

Yield information about each updated feed.

Re-raise *before_feeds_update_hooks* failures immediately. Yield feed/entry update hook failures. Collect *after_feeds_update_hooks* failures and re-raise them as an *UpdateHookErrorGroup* after updating all the feeds.

By default, update all the feeds that have updates enabled.

**Parameters**

- **feed** (*str or tuple(str) or Feed or None*) – Only update the feed with this URL.
- **tags** (*None or bool or list(str or bool or list(str or bool))*) – Only update feeds matching these tags; see *TagFilterInput* for details.
- **broken** (*bool or None*) – Only update broken / healthy feeds.
- **updates_enabled** (*bool or None*) – Only update feeds that have updates enabled / disabled. Defaults to true.
- **new** (*bool or None*) – Only update feeds that have never been updated / have been updated before. Defaults to None.
- **workers** (*int*) – Number of threads to use when getting the feeds.

**Yields**

*UpdateResult* – An (url, value) pair; the value is one of:

- a summary of the updated feed, if the update was successful
- None, if the server indicated the feed has not changed since the last update
- an exception instance

Currently, the exception can be:

- *ParseError*, if retrieving/parsing the feed failed
- *UpdateHookError*, for unexpected hook exceptions raised in *before_feed_update_hooks*, *after_entry_update_hooks*, or *after_feed_update_hooks*

...but other *UpdateError* subclasses may be yielded in the future.

**Raises**

- ***UpdateHookError*** – For unexpected hook exceptions raised in *before_feeds_update_hooks* or *after_feeds_update_hooks*.
- ***UpdateError*** – For non-feed-related update exceptions.
- ***StorageError*** –

Changed in version 3.8: Wrap unexpected update hook exceptions in *UpdateHookError*. Try to update all the feeds, don't stop after a feed/entry hook fails.

Changed in version 3.8: Document this method can raise non-feed-related *UpdateError*s (other than *UpdateHookError*).

New in version 2.6: The `feed`, `tags`, `broken`, and `updates_enabled` keyword arguments.

Changed in version 2.0: Removed the `new_only` parameter.

Changed in version 2.0: All parameters are keyword-only.

Changed in version 1.15: Update entries whenever their content changes. See *update_feeds()* for details.

New in version 1.14.

**update_feed**(*feed*, */*)

Update a single feed.

The feed will be updated even if updates are disabled for it.

Like `next(iter(reader.update_feeds_iter(feed=feed, updates_enabled=None)))[1]`, but raises the *UpdateError*, if any.

**Parameters**

**feed** (*str or tuple(str) or Feed*) – The feed URL.

**Returns**

A summary of the updated feed or None, if the server indicated the feed has not changed since the last update.

**Return type**

*UpdatedFeed* or None

**Raises**

- ***FeedNotFoundError*** –
- ***ParseError*** –
- ***UpdateHookError*** – For unexpected hook exceptions.
- ***UpdateError*** –

- *StorageError* –

Changed in version 3.8: Wrap unexpected update hook exceptions in *UpdateHookError*.

Changed in version 3.8: Document this method can raise *UpdateError*s (other than *ParseError* and *UpdateHookError*).

Changed in version 3.0: The `feed` argument is now positional-only.

Changed in version 1.15: Update entries whenever their content changes. See *update_feeds()* for details.

Changed in version 1.14: The method now returns UpdatedFeed or None instead of None.

**get_entries**(*, *feed=None*, *entry=None*, *read=None*, *important=None*, *has_enclosures=None*, *tags=None*, *feed_tags=None*, *sort='recent'*, *limit=None*, *starting_after=None*)

Get all or some of the entries.

Entries are sorted according to `sort`. Possible values:

`'recent'`

Most recent first. That is:

- by published date for entries imported on the first update (if an entry does not have *published*, *updated* is used)
- by added date for entries imported after that

This is to make sure newly imported entries appear at the top regardless of when the feed says they were published, while not having all the old entries at the top for new feeds.

---

**Note:** The algorithm for "recent" is a heuristic and may change over time.

---

Changed in version 3.1: Sort entries by added date most of the time, with the exception of those imported on the first update. Previously, entries would be sorted by added only if they were published less than 7 days ago.

`'random'`

Random order (shuffled). At at most 256 entries will be returned.

New in version 1.2.

**Parameters**

- **feed** (*str or tuple(str) or Feed or None*) – Only return the entries for this feed.
- **entry** (*tuple(str, str) or Entry or None*) – Only return the entry with this (feed URL, entry id) tuple.
- **read** (*bool or None*) – Only return (un)read entries.
- **important** (*bool or None or str*) – Only return (un)important entries. For more precise filtering, use one of the *TristateFilterInput* string filters.
- **has_enclosures** (*bool or None*) – Only return entries that (don't) have enclosures.
- **tags** (*None or bool or list(str or bool or list(str or bool))*) – Only return entries matching these tags; see *TagFilterInput* for details.
- **feed_tags** (*None or bool or list(str or bool or list(str or bool))*) – Only return entries from feeds matching these tags; see *TagFilterInput* for details.
- **sort** (*str*) – How to order entries; one of `'recent'` (default) or `'random'`.

- **limit** (*int or None*) – A limit on the number of entries to be returned; by default, all entries are returned.
- **starting_after** (*tuple(str, str) or Entry or None*) – Return entries after this entry; a cursor for use in pagination. Using `starting_after` with `sort='random'` is not supported.

**Yields**
*Entry* – Sorted according to `sort`.

**Raises**
- ***StorageError*** –
- ***EntryNotFoundError*** – If `starting_after` does not exist.

New in version 3.11: The `tags` keyword argument.

Changed in version 3.5: The `important` argument also accepts string values.

New in version 1.12: The `limit` and `starting_after` keyword arguments.

New in version 1.7: The `feed_tags` keyword argument.

New in version 1.2: The `sort` keyword argument.

**get_entry**(*entry: tuple[str, str] | EntryLike*, */*) → *Entry*

**get_entry**(*entry: tuple[str, str] | EntryLike*, *default: _T*, */*) → *Entry* | _T

Get an entry.

Like `next(iter(reader.get_entries(entry=entry)))`, but raises a custom exception instead of `StopIteration`.

**Parameters**
- **entry** (*tuple(str, str) or Entry*) – (feed URL, entry id) tuple.
- **default** – Returned if given and the entry does not exist.

**Returns**
The entry.

**Return type**
*Entry*

**Raises**
- ***EntryNotFoundError*** –
- ***StorageError*** –

Changed in version 3.0: The `entry` and `default` arguments are now positional-only.

**get_entry_counts**(*, *feed=None*, *entry=None*, *read=None*, *important=None*, *has_enclosures=None*, *tags=None*, *feed_tags=None*)

Count all or some of the entries.

**Parameters**
- **feed** (*str or tuple(str) or Feed or None*) – Only count the entries for this feed.
- **entry** (*tuple(str, str) or Entry or None*) – Only count the entry with this (feed URL, entry id) tuple.
- **read** (*bool or None*) – Only count (un)read entries.

- **important** (*bool or None or str*) – Only count (un)important entries. For more precise filtering, use one of the *TristateFilterInput* string filters.
- **has_enclosures** (*bool or None*) – Only count entries that (don't) have enclosures.
- **tags** (*None or bool or list(str or bool or list(str or bool))*) – Only count entries matching these tags; see *TagFilterInput* for details.
- **feed_tags** (*None or bool or list(str or bool or list(str or bool))*) – Only count entries from feeds matching these tags; see *TagFilterInput* for details.

**Return type**
*EntryCounts*

**Raises**
*StorageError* –

New in version 3.11: The `tags` keyword argument.

Changed in version 3.5: The `important` argument also accepts string values.

New in version 1.11.

**set_entry_read**(*entry*, *read*, */*, *modified=no value*)

Mark an entry as read or unread, possibly with a custom timestamp.

**Parameters**

- **entry** (*tuple(str, str) or Entry*) – (feed URL, entry id) tuple.
- **read** (*bool*) – Mark the entry as read if true, and as unread otherwise.
- **modified** (*datetime or None*) – Set *read_modified* to this. Naive datetimes are normalized by passing them to `astimezone()`. Defaults to the current time.

**Raises**

- *EntryNotFoundError* –
- *StorageError* –

Changed in version 3.5: Do not coerce `read` to `bool` anymore, require it to be `True` or `False`.

Changed in version 3.0: The `entry` and `read` arguments are now positional-only.

New in version 2.2.

**mark_entry_as_read**(*entry*, */*)

Mark an entry as read.

Alias for `set_entry_read(entry, True)`.

**Parameters**
**entry** (*tuple(str, str) or Entry*) – (feed URL, entry id) tuple.

**Raises**

- *EntryNotFoundError* –
- *StorageError* –

Changed in version 3.0: The `entry` argument is now positional-only.

New in version 1.18: Renamed from `mark_as_read()`.

mark_entry_as_unread(*entry*, /)

Mark an entry as unread.

Alias for `set_entry_read(entry, False)`.

**Parameters**
**entry** (*tuple(str, str) or Entry*) – (feed URL, entry id) tuple.

**Raises**

- *EntryNotFoundError* –
- *StorageError* –

Changed in version 3.0: The `entry` argument is now positional-only.

New in version 1.18: Renamed from `mark_as_unread()`.

set_entry_important(*entry*, *important*, /, *modified=no value*)

Mark an entry as important or unimportant, possibly with a custom timestamp.

**Parameters**

- **entry** (*tuple(str, str) or Entry*) – (feed URL, entry id) tuple.
- **important** (*bool or None*) – Mark the entry as important if true, as unimportant if false, or as not set if none.
- **modified** (*datetime or None*) – Set *important_modified* to this. Naive datetimes are normalized by passing them to `astimezone()`. Defaults to the current time.

**Raises**

- *EntryNotFoundError* –
- *StorageError* –

Changed in version 3.5: `important` can now be `None`.

Changed in version 3.5: Do not coerce `important` to `bool` anymore, require it to be `True` or `False` or `None`.

Changed in version 3.0: The `entry` and `important` arguments are now positional-only.

New in version 2.2.

mark_entry_as_important(*entry*, /)

Mark an entry as important.

Alias for `set_entry_important(entry, True)`.

**Parameters**
**entry** (*tuple(str, str) or Entry*) – (feed URL, entry id) tuple.

**Raises**

- *EntryNotFoundError* –
- *StorageError* –

Changed in version 3.0: The `entry` argument is now positional-only.

New in version 1.18: Renamed from `mark_as_important()`.

mark_entry_as_unimportant(*entry*, /)

Mark an entry as unimportant.

Alias for `set_entry_important(entry, False)`.

**Parameters**
**entry** (*tuple(str, str) or Entry*) – (feed URL, entry id) tuple.

**Raises**
- *EntryNotFoundError* –
- *StorageError* –

Changed in version 3.0: The `entry` argument is now positional-only.

New in version 1.18: Renamed from `mark_as_unimportant()`.

add_entry(*entry*, /)

Add a new entry to an existing feed.

`entry` can be any *Entry*-like object, or a mapping of the same shape:

```
>>> from types import SimpleNamespace
>>> reader.add_entry(SimpleNamespace(
...     feed_url='http://example.com',
...     id='one',
...     title='title',
...     enclosures=[SimpleNamespace(href='enclosure')],
... ))
>>> reader.add_entry({
...     'feed_url': 'http://example.com',
...     'id': 'two',
...     'updated': datetime.now(timezone.utc),
...     'content': [{'value': 'content'}],
... })
```

The following attributes are used (they must have the same types as on *Entry*):

- *feed_url* (required)
- *id* (required)
- *updated*
- *title*
- *link*
- *author*
- *published*
- *summary*
- *content*
- *enclosures*

Naive datetimes are normalized by passing them to `astimezone()`.

The added entry will be *added_by* `'user'`.

**Parameters**
**entry** (*Entry or dict*) – An entry-like object or equivalent mapping.

**Raises**

- *EntryExistsError* – If an entry with the same id already exists.
- *FeedNotFoundError* –
- *StorageError* –

Changed in version 3.0: The `entry` argument is now positional-only.

New in version 2.5.

**delete_entry**(*entry*, */*, *missing_ok=False*)

Delete an entry.

Currently, only entries added by *add_entry()* (*added_by* `'user'`) can be deleted.

**Parameters**

- **entry** (*tuple(str, str) or Entry*) – (feed URL, entry id) tuple.
- **missing_ok** (*bool*) – If true, don't raise *EntryNotFoundError* if the entry does not exist.

**Raises**

- *EntryNotFoundError* – If the entry does not exist, and *missing_ok* is false.
- *EntryError* – If the entry was not added by the user.
- *StorageError* –

Changed in version 3.0: The `entry` argument is now positional-only.

New in version 2.8: The `missing_ok` argument.

New in version 2.5.

**enable_search**()

Enable full-text search.

Calling this method if search is already enabled is a no-op.

**Raises**

- *SearchError* –
- *StorageError* –

**disable_search**()

Disable full-text search.

Calling this method if search is already disabled is a no-op.

**Raises**

*SearchError* –

**is_search_enabled**()

Check if full-text search is enabled.

**Returns**

Whether search is enabled or not.

**Return type**

bool

**Raises**

*SearchError* –

update_search()

Update the full-text search index.

Search must be enabled to call this method.

If *make_reader()* was called with `search_enabled='auto'` and search is disabled, it will be enabled automatically.

**Raises**

- *SearchNotEnabledError* –
- *SearchError* –
- *StorageError* –

search_entries(*query*, */*, ***, *feed=None*, *entry=None*, *read=None*, *important=None*, *has_enclosures=None*, *tags=None*, *feed_tags=None*, *sort='relevant'*, *limit=None*, *starting_after=None*)

Get entries matching a full-text search query.

Entries are sorted according to `sort`. Possible values:

`'relevant'`

Most relevant first.

`'recent'`

Most recent first. See *get_entries()* for details on what *recent* means.

Changed in version 3.1: Sort entries by added date most of the time, with the exception of those imported on the first update. Previously, entries would be sorted by added only if they were published less than 7 days ago.

New in version 1.4.

`'random'`

Random order (shuffled). At at most 256 entries will be returned.

New in version 1.10.

---

**Note:** The query syntax is dependent on the search provider.

The default (and for now, only) search provider is SQLite FTS5. You can find more details on its query syntax here: https://www.sqlite.org/fts5.html#full_text_query_syntax

The columns available in queries are:

- `title`: the entry title
- `feed`: the feed title
- `content`: the entry main text content; this includes the summary and the value of contents that have text/(x)html, text/plain or missing content types

Query examples:

- `hello internet`: entries that match "hello" and "internet"
- `hello NOT internet`: entries that match "hello" but do not match "internet"
- `hello feed: cortex`: entries that match "hello" anywhere, and their feed title matches "cortex"
- `hello NOT feed: internet`: entries that match "hello" anywhere, and their feed title does not match "internet"

Search must be enabled to call this method.

**Parameters**

- **query** (*str*) – The search query.
- **feed** (*str or tuple(str) or Feed or None*) – Only search the entries for this feed.
- **entry** (*tuple(str, str) or Entry or None*) – Only search for the entry with this (feed URL, entry id) tuple.
- **read** (*bool or None*) – Only search (un)read entries.
- **important** (*bool or None or str*) – Only search (un)important entries. For more precise filtering, use one of the *TristateFilterInput* string filters.
- **has_enclosures** (*bool or None*) – Only search entries that (don't) have enclosures.
- **tags** (*None or bool or list(str or bool or list(str or bool))*) – Only search entries matching these tags; see *TagFilterInput* for details.
- **feed_tags** (*None or bool or list(str or bool or list(str or bool))*) – Only search entries from feeds matching these tags; see *TagFilterInput* for details.
- **sort** (*str*) – How to order results; one of `'relevant'` (default), `'recent'`, or `'random'`.
- **limit** (*int or None*) – A limit on the number of results to be returned; by default, all results are returned.
- **starting_after** (*tuple(str, str) or EntrySearchResult or None*) – Return results after this result; a cursor for use in pagination. Using `starting_after` with `sort='random'` is not supported.

**Yields**

*EntrySearchResult* – Sorted according to `sort`.

**Raises**

- ***SearchNotEnabledError*** –
- ***InvalidSearchQueryError*** –
- ***SearchError*** –
- ***StorageError*** –
- ***EntryNotFoundError*** – If `starting_after` does not exist.

New in version 3.11: The `tags` keyword argument.

Changed in version 3.5: The `important` argument also accepts string values.

Changed in version 3.0: The `query` argument is now positional-only.

New in version 1.12: The `limit` and `starting_after` keyword arguments.

New in version 1.7: The `feed_tags` keyword argument.

New in version 1.4: The `sort` keyword argument.

**search_entry_counts**(*query*, */*, ***, *feed=None*, *entry=None*, *read=None*, *important=None*, *has_enclosures=None*, *tags=None*, *feed_tags=None*)

Count entries matching a full-text search query.

See *search_entries()* for details on the query syntax.

Search must be enabled to call this method.

**Parameters**

- **query** (*str*) – The search query.
- **feed** (*str or tuple(str) or Feed or None*) – Only count the entries for this feed.
- **entry** (*tuple(str, str) or Entry or None*) – Only count the entry with this (feed URL, entry id) tuple.
- **read** (*bool or None or str*) – Only count (un)read entries. For more precise filtering, use one of the *TristateFilterInput* string filters.
- **important** (*bool or None*) – Only count (un)important entries.
- **has_enclosures** (*bool or None*) – Only count entries that (don't) have enclosures.
- **tags** (*None or bool or list(str or bool or list(str or bool))*) – Only count entries matching these tags; see *TagFilterInput* for details.
- **feed_tags** (*None or bool or list(str or bool or list(str or bool))*) – Only count entries from feeds matching these tags; see *TagFilterInput* for details.

**Return type**
*EntrySearchCounts*

**Raises**

- ***SearchNotEnabledError*** –
- ***InvalidSearchQueryError*** –
- ***SearchError*** –
- ***StorageError*** –

New in version 3.11: The `tags` keyword argument.

Changed in version 3.5: The `important` argument also accepts string values.

Changed in version 3.0: The `query` argument is now positional-only.

New in version 1.11.

**get_tags**(*resource*, */*, ***, *key=None*)

Get all or some tags of a resource as `(key, value)` pairs.

*resource* can have one of the following types:

*Feed* or `str` or `(str,)`

A feed or feed URL (possibly enclosed in a tuple).

*Entry* or `(str, str)`

An entry or a (feed URL, entry id) pair representing an entry.

`()` (empty tuple)

Special value representing the global tag namespace.

**Parameters**

- **resource** (*reader.types.ResourceInput*) – The resource to get tags for.
- **key** (*str or None*) – Only return the value for this key.

**Yields**
*tuple(str, JSONType)* – `(key, value)` pairs, in undefined order. JSONType is whatever `json.dumps()` accepts.

**Raises**
*StorageError* –

Changed in version 3.0: The `resource` argument is now positional-only.

Changed in version 2.10: Support entry and global tags.

Changed in version 2.10: Removed support for the `(None,)` (any feed) and `None` (any resource) wildcard resource values.

New in version 2.8.

**get_tag_keys**(*resource=None*, /)

Get the keys of all or some resource tags.

Equivalent to `sorted(k for k, _ in reader.get_tags(resource))`.

See *get_tags()* for possible *resource* values. In addition, *resource* can have one of the following wildcard values:

`(None,)`

Any feed.

`(None, None)`

Any entry.

`None`

Any resource (feed, entry, or the global namespace).

**Parameters**
**resource** (*reader.types.AnyResourceInput*) – Only return tag keys for this resource.

**Yields**
*str* – The tag keys, in alphabetical order.

**Raises**
*StorageError* –

Changed in version 3.0: The `resource` argument is now positional-only.

Changed in version 2.10: Support entry and global tags.

New in version 2.8.

**get_tag**(*resource: ResourceInput*, *key: str*, /) → JSONType

**get_tag**(*resource: ResourceInput*, *key: str*, *default: _T*, /) → reader.types.JSONType | _T

Get the value of this resource tag.

Like `next(iter(reader.get_tags(resource, key=key)))[1]`, but raises a custom exception instead of `StopIteration`.

See *get_tags()* for possible *resource* values.

**Parameters**

- **resource** – The resource.
- **key** (*str*) – The key of the tag to retrieve.

- **default** – Returned if given and no tag exists for *key*.

**Returns**
The tag value. JSONType is whatever `json.dumps()` accepts.

**Return type**
JSONType

**Raises**

- *TagNotFoundError* –
- *StorageError* –

Changed in version 3.0: The `resource`, `key`, and `default` arguments are now positional-only.

Changed in version 2.10: Support entry and global tags.

New in version 2.8.

**set_tag**(*resource: ResourceInput*, *key: str*, */*) → None

**set_tag**(*resource: ResourceInput*, *key: str*, *value: JSONType*, */*) → None

Set the value of this resource tag.

See *get_tags()* for possible *resource* values.

**Parameters**

- **resource** – The resource.
- **key** (*str*) – The key of the tag to set.
- **value** (*JSONType*) – The value of the tag to set. If not provided, and the tag already exists, the value remains unchanged; if the tag does not exist, it is set to `None`. JSONType is whatever `json.dumps()` accepts.

**Raises**

- *ResourceNotFoundError* –
- *StorageError* –

Changed in version 3.0: The `resource`, `key`, and `value` arguments are now positional-only.

Changed in version 2.10: Support entry and global tags.

New in version 2.8.

**delete_tag**(*resource*, *key*, */*, *missing_ok=False*)

Delete this resource tag.

See *get_tags()* for possible *resource* values.

**Parameters**

- **resource** (*reader.types.ResourceInput*) – The resource.
- **key** (*str*) – The key of the tag to delete.
- **missing_ok** (*bool*) – If true, don't raise *TagNotFoundError* if the tag does not exist.

**Raises**

- *TagNotFoundError* – If the tag does not exist, and *missing_ok* is false.
- *StorageError* –

Changed in version 3.0: The `resource` and `key` arguments are now positional-only.

Changed in version 2.10: Support entry and global tags.

New in version 2.8.

**make_reader_reserved_name**(*key*, */*)

Create a *reader*-reserved tag name. See *Reserved names* for details.

Uses *reserved_name_scheme* to build names of the format:

```
{reader_prefix}{key}
```

Using the default scheme:

```
>>> reader.make_reader_reserved_name('key')
'.reader.key'
```

**Parameters**
**key** (*str*) – A key.

**Returns**
The name.

**Return type**
str

Changed in version 3.0: The `key` argument is now positional-only.

New in version 1.17.

**make_plugin_reserved_name**(*plugin_name*, *key=None*, */*)

Create a plugin-reserved tag name. See *Reserved names* for details.

Plugins should use this to generate names for plugin-specific tags.

Uses *reserved_name_scheme* to build names of the format:

```
{plugin_prefix}{plugin_name}
{plugin_prefix}{plugin_name}{separator}{key}
```

Using the default scheme:

```
>>> reader.make_plugin_reserved_name('myplugin')
'.plugin.myplugin'
>>> reader.make_plugin_reserved_name('myplugin', 'key')
'.plugin.myplugin.key'
```

**Parameters**

- **plugin_name** (*str*) – The plugin package/module name.
- **key** (*str or None*) – A key; if more than one reserved name is needed.

**Returns**
The name.

**Return type**
str

Changed in version 3.0: The `plugin_name` and `key` arguments are now positional-only.

New in version 1.17.

**property reserved_name_scheme: Mapping[str, str]**

Mapping used to build reserved names. See *make_reader_reserved_name()* and *make_plugin_reserved_name()* for details on how this is used.

The default scheme (these keys are required):

```
{'reader_prefix': '.reader.', 'plugin_prefix': '.plugin.', 'separator': '.'}
```

The returned mapping is immutable; assign a new mapping to change the scheme.

New in version 1.17.

**Type**
dict(str, str)

**property before_feeds_update_hooks: MutableSequence[Callable[[*Reader*], None]]**

List of functions called *once* before updating any feeds, at the beginning of *update_feeds()* / *update_feeds_iter()*, but not *update_feed()*.

Each function is called with:

- *reader* – the *Reader* instance

Each function should return `None`.

The hooks are run in order. Exceptions raised by hooks are wrapped in a *SingleUpdateHookError* and re-raised (hooks after the one that failed are not run).

Changed in version 3.8: Wrap unexpected exceptions in *UpdateHookError*.

New in version 2.12.

**property before_feed_update_hooks: MutableSequence[Callable[[*Reader*, str], None]]**

List of functions called for each updated feed before the feed is updated.

Each function is called with:

- *reader* – the *Reader* instance
- *feed* – the `str` feed URL

Each function should return `None`.

The hooks are run in order. Exceptions raised by hooks are wrapped in a *SingleUpdateHookError* and re-raised (hooks after the one that failed are not run).

Changed in version 3.8: Wrap unexpected exceptions in *UpdateHookError*.

New in version 2.7.

**property after_entry_update_hooks: MutableSequence[Callable[[*Reader*, *EntryData*, *EntryUpdateStatus*], None]]**

List of functions called for each updated entry after the feed is updated.

Each function is called with:

- *reader* – the *Reader* instance
- *entry* – an *Entry*-like object
- *status* – an *EntryUpdateStatus* value

Each function should return None.

> **Warning:** The only *entry* attributes guaranteed to be present are *feed_url*, *id*, and *resource_id*; all other attributes may be missing (accessing them may raise AttributeError).

The hooks are run in order. Exceptions raised by hooks are wrapped in a *SingleUpdateHookError*, collected, and re-raised as an *UpdateHookErrorGroup* after all the hooks are run; currently, only the exceptions for the first 5 entries with hook failures are collected.

Changed in version 3.8: Wrap unexpected exceptions in *UpdateHookError*. Try to run all hooks, don't stop after one fails.

New in version 1.20.

**property after_feed_update_hooks:** MutableSequence[Callable[[*Reader*, str], None]]

List of functions called for each updated feed after the feed is updated.

Each function is called with:

- *reader* – the *Reader* instance
- *feed* – the str feed URL

Each function should return None.

The hooks are run in order. Exceptions raised by hooks are wrapped in a *SingleUpdateHookError*, collected, and re-raised as an *UpdateHookErrorGroup* after all the hooks are run.

Changed in version 3.8: Wrap unexpected exceptions in *UpdateHookError*. Try to run all hooks, don't stop after one fails.

New in version 2.2.

**property after_feeds_update_hooks:** MutableSequence[Callable[[*Reader*], None]]

List of functions called *once* after updating all feeds, at the end of *update_feeds()* / *update_feeds_iter()*, but not *update_feed()*.

Each function is called with:

- *reader* – the *Reader* instance

Each function should return None.

The hooks are run in order. Exceptions raised by hooks are wrapped in a *SingleUpdateHookError*, collected, and re-raised as an *UpdateHookErrorGroup* after all the hooks are run.

Changed in version 3.8: Wrap unexpected exceptions in *UpdateHookError*. Try to run all hooks, don't stop after one fails.

New in version 2.12.

### 4.1.2 Data objects

class reader.**Feed**(*url*, *updated=None*, *title=None*, *link=None*, *author=None*, *subtitle=None*, *version=None*, *user_title=None*, *added=None*, *last_updated=None*, *last_exception=None*, *updates_enabled=True*)

Data type representing a feed.

All `datetime` attributes are timezone-aware, with the timezone set to `utc`.

Changed in version 2.0: `datetime` attributes are now timezone-aware; prior to 2.0, they were naive datetimes representing UTC times.

**url:** `str`

The URL of the feed.

**updated:** `datetime | None = None`

The date the feed was last updated, according to the feed.

**title:** `str | None = None`

The title of the feed.

**link:** `str | None = None`

The URL of a page associated with the feed.

**author:** `str | None = None`

The author of the feed.

**subtitle:** `str | None = None`

A description or subtitle for the feed.

New in version 2.4.

**version:** `str | None = None`

The feed type and version.

For Atom and RSS, provided by feedparser (e.g. `atom10`, `rss20`); full list.

For JSON Feed:

**json10**
: JSON Feed 1.0

**json11**
: JSON Feed 1.1

**json**
: JSON Feed (unknown or unrecognized version)

Plugins may add other versions.

New in version 2.4.

**user_title:** `str | None = None`

User-defined feed title.

**added:** `datetime = None`

The date when the feed was added.

New in version 1.3.

**last_updated:** *datetime* | None = None

The date when the feed was last retrieved by reader.

New in version 1.3.

**last_exception:** *ExceptionInfo* | None = None

If a *UpdateError* happened during the last update, its details.

Changed in version 3.9: Store the details of any *UpdateError* (except hook errors), not just the `__cause__` of *ParseError*s.

New in version 1.3.

**updates_enabled:** bool = True

Whether updates are enabled for this feed.

New in version 1.11.

**property resource_id:** tuple[str]

Alias for (*url*,).

New in version 2.17.

class reader.**ExceptionInfo**(*type_name*, *value_str*, *traceback_str*)

Data type representing information about an exception.

New in version 1.3.

**type_name:** str

The fully qualified name of the exception type.

**value_str:** str

String representation of the exception value.

**traceback_str:** str

String representation of the exception traceback.

class reader.**Entry**(*id*, *updated=None*, *title=None*, *link=None*, *author=None*, *published=None*, *summary=None*, *content=()*, *enclosures=()*, *read=False*, *read_modified=None*, *important=None*, *important_modified=None*, *added=None*, *added_by=None*, *last_updated=None*, *original_feed_url=None*, *_sequence=None*, *feed=None*)

Data type representing an entry.

All `datetime` attributes are timezone-aware, with the timezone set to `utc`.

Changed in version 2.0: `datetime` attributes are now timezone-aware; prior to 2.0, they were naive datetimes representing UTC times.

**property feed_url:** str

The feed URL.

**id:** str

The entry id.

**updated:** datetime | None = None

The date the entry was last updated, according to the feed.

Changed in version 2.5: Is now None if missing in the feed; use *updated_not_none* for the pre-2.5 behavior.

Changed in version 2.0: May be None in some cases. In a future version, will be None if missing in the feed; use *updated_not_none* for the pre-2.0 behavior.

**title:** str | None = None

The title of the entry.

**link:** str | None = None

The URL of a page associated with the entry.

**author:** str | None = None

The author of the feed.

**published:** datetime | None = None

The date the entry was published, according to the feed.

**summary:** str | None = None

A summary of the entry.

**content:** Sequence[*Content*] = ()

Full content of the entry. A sequence of *Content* objects.

**enclosures:** Sequence[*Enclosure*] = ()

External files associated with the entry. A sequence of *Enclosure* objects.

**read:** bool = False

Whether the entry was read or not.

**read_modified:** datetime | None = None

The date when *read* was last set by the user; None if that never happened, or the entry predates the date being recorded.

New in version 2.2.

**important:** bool | None = None

Whether the entry is important or not. None means not set. False means "explicitly unimportant".

Changed in version 3.5: *important* is now an optional bool, and defaults to None.

**important_modified:** datetime | None = None

The date when *important* was last set by the user; None if that never happened, or the entry predates the date being recorded.

New in version 2.2.

**added:** datetime = None

The date when the entry was added (first updated) to reader.

New in version 2.5.

**added_by:** Literal['feed', 'user'] = None

The source of the entry. One of 'feed', 'user'.

Other values may be added in the future.

New in version 2.5.

**last_updated:** *datetime* = None

The date when the entry was last updated by reader.

New in version 1.3.

**original_feed_url:** *str* = None

The URL of the original feed of the entry.

If the feed URL never changed, the same as *feed_url*.

New in version 1.8.

**feed:** *Feed* = None

The entry's feed.

**property resource_id:** *tuple*[*str*, *str*]

Alias for (*feed_url*, *id*).

New in version 2.17.

**property updated_not_none:** *datetime*

Like *updated*, but guaranteed to be set (not None).

If the entry *updated* is missing in the feed, defaults to when the entry was first *added*.

New in version 2.0: Identical to the behavior of *updated* before 2.0.

**get_content**(*, *prefer_summary=False*)

Return a text content OR the summary.

Prefer HTML content, when available.

**Parameters**
**prefer_summary** (*bool*) – Return summary, if available.

**Returns**
The content, if found.

**Return type**
*Content* or none

New in version 2.12.

**class** reader.**Content**(*value*, *type=None*, *language=None*)

Data type representing a piece of content.

**value:** *str*

The content value.

**type:** *str* | None = None

The content type.

**language:** *str* | None = None

The content language.

**property is_html:** *bool*

Whether the content is (X)HTML.

True if the content does not have a type.

New in version 2.12.

class reader.**Enclosure**(*href*, *type=None*, *length=None*)

Data type representing an external file.

**href:** str

The file URL.

**type:** str | None = None

The file content type.

**length:** int | None = None

The file length.

class reader.**EntrySearchResult**(*feed_url*, *id*, *metadata=<factory>*, *content=<factory>*)

Data type representing the result of an entry search.

*metadata* and *content* are dicts where the key is the path of an entry attribute, and the value is a *HighlightedString* snippet corresponding to that attribute, with HTML stripped.

```
>>> result = next(reader.search_entries('hello internet'))
>>> result.metadata['.title'].value
'A Recent Hello Internet'
>>> reader.get_entry(result).title
'A Recent Hello Internet'
```

**feed_url:** str

The feed URL.

**id:** str

The entry id.

**metadata:** Mapping[str, *HighlightedString*]

Matching entry metadata, in arbitrary order. Currently entry.title and entry.feed.user_title/.title.

**content:** Mapping[str, *HighlightedString*]

Matching entry content, sorted by relevance. Any of entry.summary and entry.content[].value.

**property resource_id:** tuple[str, str]

Alias for (*feed_url*, *id*).

New in version 2.17.

class reader.**HighlightedString**(*value=''*, *highlights=()*)

A string that has some of its parts highlighted.

**value:** str = ''

The underlying string.

**highlights:** Sequence[slice] = ()

The highlights; non-overlapping slices with positive start/stop and None step.

**classmethod extract**(*text*, *before*, *after*)

Extract highlights with before/after markers from text.

```
>>> HighlightedString.extract( '>one< two', '>', '<')
HighlightedString(value='one two', highlights=(slice(0, 3, None),))
```

**Parameters**

- **text** (*str*) – The original text, with highlights marked by `before` and `after`.
- **before** (*str*) – Highlight start marker.
- **after** (*str*) – Highlight stop marker.

**Returns**
A highlighted string.

**Return type**
*HighlightedString*

**split**()

Split the highlighted string into parts.

```
>>> list(HighlightedString('abcd', [slice(1, 3)]))
['a', 'bc', 'd']
```

**Yields**
*str* – The parts (always an odd number); parts with odd indexes are highlighted, parts with even indexes are not.

**apply**(*before*, *after*, *func=None*)

Apply before/end markers on the highlighted string.

The opposite of *extract()*.

```
>>> HighlightedString('abcd', [slice(1, 3)]).apply('>', '<')
'a>bc<d'
>>> HighlightedString('abcd', [slice(1, 3)]).apply('>', '<', str.upper)
'A>BC<D'
```

**Parameters**

- **before** (*str*) – Highlight start marker.
- **after** (*str*) – Highlight stop marker.
- **func** (*callable((str), str) or none*) – If given, a function to apply to the string parts before adding the markers.

**Returns**
The string, with highlights marked by `before` and `after`.

**Return type**
str

class reader.**FeedCounts**(*total=None*, *broken=None*, *updates_enabled=None*)

Count information about feeds.

New in version 1.11.

**total:** int | None = None

Total number of feeds.

**broken:** int | None = None

Number of broken feeds.

**updates_enabled:** int | None = None

Number of feeds that have updates enabled.

class reader.**EntryCounts**(*total=None*, *read=None*, *important=None*, *has_enclosures=None*, *averages=None*)

Count information about entries.

New in version 1.11.

**total:** int | None = None

Total number of entries.

**read:** int | None = None

Number of read entries.

**important:** int | None = None

Number of important entries.

**has_enclosures:** int | None = None

Number of entries that have enclosures.

**averages:** tuple[float, float, float] | None = None

Average entries per day during the last 1, 3, 12 months, as a 3-tuple.

New in version 2.1.

class reader.**EntrySearchCounts**(*total=None*, *read=None*, *important=None*, *has_enclosures=None*, *averages=None*)

Count information about entry search results.

New in version 1.11.

**total:** int | None = None

Total number of entries.

**read:** int | None = None

Number of read entries.

**important:** int | None = None

Number of important entries.

**has_enclosures:** int | None = None

Number of entries that have enclosures.

**averages:** tuple[float, float, float] | None = None

Average entries per day during the last 1, 3, 12 months, as a 3-tuple.

New in version 2.1.

class reader.**UpdateResult**(*url*, *value*)

Named tuple representing the result of a feed update.

New in version 1.14.

**url:** str

The URL of the feed.

**value:** *UpdatedFeed* | None | *UpdateError*

One of:

*UpdatedFeed*

If the update was successful; a summary of the updated feed.

None

If the server indicated the feed has not changed since the last update without returning any data.

*UpdateError*

If there was an error while updating the feed.

Changed in version 3.8: Narrow down the error type from *ReaderError* to *UpdateError*.

**property updated_feed:** *UpdatedFeed* | None

The updated feed, if the update was successful, None otherwise.

New in version 2.1.

**property error:** *UpdateError* | None

The exception, if there was an error, None otherwise.

New in version 2.1.

**property not_modified:** bool

True if the feed has not changed (either because the server returned no data, or because the data didn't change), false otherwise.

New in version 2.1.

class reader.**UpdatedFeed**(*url*, *new=0*, *modified=0*, *unmodified=0*)

The result of a successful feed update.

Changed in version 1.19: The `updated` argument/attribute was renamed to `modified`.

New in version 1.14.

**url:** str

The URL of the feed.

**new:** int = 0

The number of new entries (entries that did not previously exist in storage).

Changed in version 3.2: This field is now optional, and defaults to 0.

**modified:** int = 0

The number of modified entries (entries that existed in storage, but had different data than the corresponding feed file entry.)

Changed in version 3.2: This field is now optional, and defaults to 0.

**unmodified:** int = 0

The number of unmodified entries (entries that existed in storage, but had the same data in the corresponding feed file entry.)

New in version 3.2.

**property total:** int

The total number of entries in the retrieved feed.

New in version 3.2.

**class** reader.**EntryUpdateStatus**(*value*, *names=None*, **values*, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

Enum representing how an entry was updated.

New in version 1.20.

**NEW = 'new'**

The entry did not previously exist in storage.

**MODIFIED = 'modified'**

The entry existed in storage, but had different data from the one in the feed file.

### 4.1.3 Exceptions

**exception** reader.**ReaderError**(*message=''*)

Base for all public exceptions.

**exception** reader.**FeedError**(*url*, */*, *message=''*)

Bases: *ReaderError*

A feed error occurred.

Changed in version 3.0: The `url` argument is now positional-only.

**property resource_id:** tuple[str]

Alias for (`url`,).

New in version 2.17.

**exception** reader.**FeedExistsError**(*url*, */*, *message=''*)

Bases: *FeedError*

Feed already exists.

exception reader.FeedNotFoundError(*url*, /, *message=''*)

Bases: *FeedError*, *ResourceNotFoundError*

Feed not found.

exception reader.InvalidFeedURLError(*url*, /, *message=''*)

Bases: *FeedError*, ValueError

Invalid feed URL.

New in version 2.5.

exception reader.EntryError(*feed_url*, *id*, /, *message=''*)

Bases: *ReaderError*

An entry error occurred.

Changed in version 3.0: The `feed_url` and `id` arguments are now positional-only.

Changed in version 1.18: The `url` argument/attribute was renamed to `feed_url`.

property resource_id: tuple[str, str]

Alias for (`feed_url`, `id`).

New in version 2.17.

exception reader.EntryExistsError(*feed_url*, *id*, /, *message=''*)

Bases: *EntryError*

Entry already exists.

New in version 2.5.

exception reader.EntryNotFoundError(*feed_url*, *id*, /, *message=''*)

Bases: *EntryError*, *ResourceNotFoundError*

Entry not found.

exception reader.UpdateError(*message=''*)

Bases: *ReaderError*

An error occurred while updating the feed.

Parent of all update-related exceptions.

New in version 3.8.

exception reader.ParseError(*url*, /, *message=''*)

Bases: *UpdateError*, *FeedError*, *ReaderWarning*

An error occurred while retrieving/parsing the feed.

The original exception should be chained to this one (e.__cause__).

Changed in version 3.8: Inherit from *UpdateError*.

exception reader.UpdateHookError(*message=''*)

Bases: *UpdateError*

One or more update hooks (unexpectedly) failed.

Not raised directly; allows catching any hook errors with a single except clause.

To inspect individual hook failures, use except* with *SingleUpdateHookError* (or, on Python earlier than 3.11, check if the exception `isinstance()` *UpdateHookErrorGroup* and examine its `exceptions`).

New in version 3.8.

exception reader.SingleUpdateHookError(*when*, *hook*, *resource_id=None*)

Bases: *UpdateHookError*

An update hook (unexpectedly) failed.

The original exception should be chained to this one (e.__cause__).

New in version 3.8.

**when**

The update phase (the hook type). One of:

- `'before_feeds_update'`
- `'before_feed_update'`
- `'after_entry_update'`
- `'after_feed_update'`
- `'after_feeds_update'`

**hook**

The hook.

**resource_id**

The *resource_id* of the resource, if any.

exception reader.UpdateHookErrorGroup(*msg*, *excs*, */*)

Bases: `ExceptionGroup`, *UpdateHookError*

A (possibly nested) `ExceptionGroup` of *UpdateHookError*s.

New in version 3.8.

exception reader.StorageError(*message=''*)

Bases: *ReaderError*

An exception was raised by the underlying storage.

The original exception should be chained to this one (e.__cause__).

exception reader.SearchError(*message=''*)

Bases: *ReaderError*

A search-related exception.

If caused by an exception raised by the underlying search provider, the original exception should be chained to this one (e.__cause__).

exception reader.SearchNotEnabledError(*message=''*)

Bases: *SearchError*

A search-related method was called when search was not enabled.

exception reader.InvalidSearchQueryError(*message=''*)

Bases: *SearchError*, ValueError

The search query provided was somehow invalid.

exception reader.TagError(*resource_id*, *key*, */*, *message=''*)

Bases: *ReaderError*

A tag error occurred.

Changed in version 3.0: Signature changed from `TagError(key, resource_id, ...)` to `TagError(resource_id, key, ...)`.

Changed in version 3.0: The `resource_id` and `key` arguments are now positional-only.

Changed in version 2.17: Signature changed from `TagError(key, object_id, ...)` to `TagError(key, resource_id, ...)`.

New in version 2.8.

exception reader.TagNotFoundError(*resource_id*, *key*, */*, *message=''*)

Bases: *TagError*

Tag not found.

New in version 2.8.

exception reader.ResourceNotFoundError(*message=''*)

Bases: *ReaderError*

Resource (feed, entry) not found.

New in version 2.8.

property resource_id: tuple[str, ...]

The *resource_id* of the resource.

**exception** reader.**PluginError**(*message=''*)

Bases: *ReaderError*

A plugin-related exception.

**exception** reader.**InvalidPluginError**(*message=''*)

Bases: *PluginError*, ValueError

An invalid plugin was provided.

New in version 1.16.

**exception** reader.**PluginInitError**(*message=''*)

Bases: *PluginError*

A plugin failed to initialize.

The original exception should be chained to this one (e.__cause__).

New in version 3.0.

**exception** reader.**ReaderWarning**(*message=''*)

Bases: *ReaderError*, UserWarning

Base for all warnings emitted by *reader* that are not DeprecationWarning.

Changed in version 3.8: Inherit from *ReaderError*.

New in version 2.13.

### Exception hierarchy

The class hierarchy for *reader* exceptions is:

```
ReaderError
  ├── ReaderWarning [UserWarning]
  ├── ResourceNotFoundError
  ├── FeedError
  │     ├── FeedExistsError
  │     ├── FeedNotFoundError [ResourceNotFoundError]
  │     └── InvalidFeedURLError [ValueError]
  ├── EntryError
  │     ├── EntryExistsError
  │     └── EntryNotFoundError [ResourceNotFoundError]
  ├── UpdateError
  │     ├── ParseError [FeedError, ReaderWarning]
  │     └── UpdateHookError
  │           ├── SingleUpdateHookError
  │           └── UpdateHookErrorGroup [ExceptionGroup]
  ├── StorageError
  │     └── ChangeTrackingNotEnabledError
  ├── SearchError
  │
```

(continues on next page)

(continued from previous page)

```
    ├── SearchNotEnabledError
    └── InvalidSearchQueryError [ValueError]
├── PluginError
    ├── InvalidPluginError [ValueError]
    └── PluginInitError
└── TagError
    └── TagNotFoundError
```

### 4.1.4 Type aliases

reader.types.**TagFilterInput**

Possible values for filtering resources by their tags.

Tag filters consist of a list of one or more tags. Multiple tags are interpreted as a conjunction (AND). To use a disjunction (OR), use a nested list. To negate a tag, prefix the tag value with a minus sign (`-`). Examples:

`['one']`

one

`['one', 'two']` `[['one'], ['two']]`

one AND two

`[['one', 'two']]`

one OR two

`[['one', 'two'], 'three']`

(one OR two) AND three

`['one', '-two']`

one AND NOT two

Special values `True` and `False` match resources with any tags and no tags, respectively.

`True` `[True]`

*any tags*

`False` `[False]`

*no tags*

`[True, '-one']`

*any tags* AND NOT one

`[[False, 'one']]`

*no tags* OR one

New in version 3.11.

alias of `Union`[`None`, `bool`, `Sequence`[`Union`[`str`, `bool`, `Sequence`[`Union`[`str`, `bool`]]]]]

reader.types.**TristateFilterInput**

Possible values for options that filter items by an optional boolean attribute (one that can be either true, false, or not set).

None selects all items. True and False select items based of the attribute's truth value (a None attribute is treated as false).

For more precise filtering, use one of the following string filters:

| attribute values | string filter | optional bool filter |
|---|---|---|
| True | istrue | True |
| False | isfalse | |
| None | notset | |
| False, None | nottrue | False |
| True, None | notfalse | |
| True, False | isset | |
| True, False, None | any | None |

New in version 3.5.

alias of Literal[None, True, False, 'istrue', 'isfalse', 'notset', 'nottrue', 'notfalse', 'isset', 'any']

### 4.1.5 Constants

reader.plugins.**DEFAULT_PLUGINS = ['reader.ua_fallback']**

The list of plugins *make_reader()* uses by default.

## 4.2 Internal API

This part of the documentation covers the internal interfaces of *reader*, which are useful for plugins, or if you want to use low-level functionality without using *Reader* itself.

**Warning:** As of version 3.12, the internal API is **not** part of the public API; it is not stable yet and might change without any notice.

### 4.2.1 Parser

Reader.**_parser**

The *Parser* instance used by this reader.

reader._parser.**default_parser**(*feed_root=None*, *session_timeout=(3.05, 60)*, *_lazy=True*)

Create a pre-configured *Parser*.

**Parameters**

- **feed_root** (*str or None*) – See *make_reader()* for details.
- **session_timeout** (*float or tuple(float, float) or None*) – See *make_reader()* for details.

**Returns**

The parser.

**Return type**

*Parser*

**class** reader._parser.**Parser**

Retrieve and parse feeds by delegating to *retrievers* and *parsers*.

To retrieve and parse a single feed, you can *call* the parser object directly.

*Reader* only uses the following methods:

- *parallel()*
- *validate_url()*
- *process_feed_for_update()*
- *process_entry_pairs()*

To add retrievers and parsers:

- *mount_retriever()*
- *mount_parser_by_mime_type()*
- *mount_parser_by_url()*

The rest of the methods are low-level methods.

**session_factory**

SessionFactory used to create Requests sessions for retrieving feeds.

Plugins may add request or response hooks to this.

**parallel**(*feeds*, *map=<class 'map'>*, *is_parallel=True*)

Retrieve and parse many feeds, possibly in parallel.

Yields the parsed feeds, as soon as they are ready.

**Parameters**

- **feeds** (*iterable(FeedArgument)*) – An iterable of feeds.
- **map** (*function*) – A map()-like function; the results can be in any order.
- **is_parallel** (*bool*) – Whether map runs the tasks in parallel.

**Yields**

tuple(*FeedArgument*, *ParsedFeed* or None or *ParseError*) – A (feed, result) pair, where result is either:

- the parsed feed
- None, if the feed didn't change
- an exception instance

**__call__**(*url*, *http_etag=None*, *http_last_modified=None*)

Retrieve and parse one feed.

This is a convenience wrapper over *parallel()*.

**Parameters**

- **feed** (*str*) – The feed URL.
- **http_etag** (*str or None*) – The HTTP ETag header from the last update.
- **http_last_modified** (*str or None*) – The the HTTP Last-Modified header from the last update.

**Returns**
The parsed feed or `None`, if the feed didn't change.

**Return type**
*ParsedFeed* or None

**Raises**
**ParseError** –

**retrieve**(*url*, *http_etag=None*, *http_last_modified=None*, *is_parallel=False*)

Retrieve a feed.

**Parameters**

- **url** (*str*) – The feed URL.
- **http_etag** (*str or None*) – The HTTP `ETag` header from the last update.
- **http_last_modified** (*str or None*) – The the HTTP `Last-Modified` header from the last update.
- **is_parallel** (*bool*) – Whether this was called from *parallel()* (writes the contents to a temporary file, if possible).

**Returns**
A context manager that has as target either the result or `None`, if the feed didn't change.

**Return type**
contextmanager(*RetrieveResult* or None)

**Raises**
**ParseError** –

**parse**(*url*, *result*)

Parse a retrieved feed.

**Parameters**

- **url** (*str*) – The feed URL.
- **result** (`RetrieveResult`) – A retrieve result.

**Returns**
The feed and entry data.

**Return type**
*ParsedFeed*

**Raises**
**ParseError** –

**get_parser**(*url*, *mime_type*)

Select an appropriate parser for a feed.

Parsers *registered by URL* take precedence over those *registered by MIME type*.

If no MIME type is given, guess it from the URL using `mimetypes.guess_type()`. If the MIME type can't be guessed, default to `application/octet-stream`.

**Parameters**

- **url** (*str*) – The feed URL.
- **mime_type** (*str or None*) – The MIME type of the retrieved resource.

**Returns**
The parser, and the (possibly guessed) MIME type.

**Return type**
tuple(*ParserType*, str)

**Raises**
**ParseError** – No parser matches.

**validate_url**(*url*)

Check if `url` is valid without actually retrieving it.

**Raises**
**InvalidFeedURLError** – If `url` is not valid.

**mount_retriever**(*prefix*, *retriever*)

Register a retriever to a URL prefix.

Retrievers are sorted in descending order by prefix length.

**Parameters**

- **prefix** (*str*) – A URL prefix.
- **retriever** (RetrieverType) – The retriever.

**get_retriever**(*url*)

Get the retriever for a URL.

**Parameters**
**url** (*str*) – The URL.

**Returns**
The matching retriever.

**Return type**
*RetrieverType*

**Raises**
**ParseError** – No retriever matches the URL.

**mount_parser_by_mime_type**(*parser*, *http_accept=None*)

Register a parser to one or more MIME types.

**Parameters**

- **parser** (ParserType) – The parser.
- **http_accept** (*str or None*) – The content types the parser supports, as an `Accept` HTTP header value. If not given, use the parser's *http_accept* attribute, if it has one.

**Raises**
**TypeError** – The parser does not have an *http_accept* attribute, and no `http_accept` was given.

**get_parser_by_mime_type**(*mime_type*)

Get a parser for a MIME type.

**Parameters**
**mime_type** (*str*) – The MIME type of the feed resource.

**Returns**
The parser.

**Return type**
*ParserType*

**Raises**
**ParseError** – No parser matches the MIME type.

**mount_parser_by_url**(*url*, *parser*)

Register a parser to an exact URL.

**Parameters**

- **prefix** (*str*) – A URL.
- **parser** (ParserType) – The parser.

**get_parser_by_url**(*url*)

Get a parser that was registered by URL.

**Parameters**
**url** (*str*) – The URL.

**Returns**
The parser.

**Return type**
*ParserType*

**Raises**
**ParseError** – No parser was registered for the URL.

**process_feed_for_update**(*feed*)

Change update-relevant information about a feed before it is passed to the retriever.

Delegates to *process_feed_for_update()* of the appropriate retriever.

**Parameters**
**feed** (FeedForUpdate) – Feed information.

**Returns**
The passed-in feed information, possibly modified.

**Return type**
*FeedForUpdate*

**process_entry_pairs**(*url*, *mime_type*, *pairs*)

Process entry data before being stored.

Delegates to *process_entry_pairs()* of the appropriate parser.

**Parameters**

- **url** (*str*) – The feed URL.
- **mime_type** (*str or None*) – The MIME type of the feed.
- **pairs** (*iterable(tuple(EntryData, EntryForUpdate or None))*) – (entry data, entry for update) pairs.

**Returns**
(entry data, entry for update) pairs, possibly modified.

**Return type**
iterable(tuple(*EntryData*, *EntryForUpdate* or None))

class reader._parser.requests.**SessionFactory**(...)

Manage the lifetime of a session.

To get new session, *call* the factory directly.

**request_hooks: Sequence[*RequestHook*]**

Sequence of *RequestHook*s to be associated with new sessions.

**response_hooks: Sequence[*ResponseHook*]**

Sequence of *ResponseHook*s to be associated with new sessions.

**__call__**()

Create a new session.

**Return type**
*SessionWrapper*

**transient**()

Return the current *persistent()* session, or a new one.

If a new session was created, it is closed once the context manager is exited.

**Return type**
contextmanager(*SessionWrapper*)

**persistent**()

Register a persistent session with this factory.

While the context manager returned by this method is entered, all *persistent()* and *transient()* calls will return the same session. The session is closed once the outermost *persistent()* context manager is exited.

Plugins should use *transient()*.

Reentrant, but NOT threadsafe.

**Return type**
contextmanager(*SessionWrapper*)

class reader._parser.requests.**SessionWrapper**(...)

Minimal wrapper over a `requests.Session`.

Only provides a limited *get()* method.

Can be used as a context manager (closes the session on exit).

**session: `requests.Session`**

The underlying `requests.Session`.

**request_hooks: Sequence[*RequestHook*]**

Sequence of *RequestHook*s.

**response_hooks: Sequence[*ResponseHook*]**

Sequence of *ResponseHook*s.

**get**(*url*, *headers=None*, ***kwargs*)

Like Requests `get()`, but apply *request_hooks* and *response_hooks*.

**Parameters**

- **url** (*str*) – Passed to `Request`.
- **headers** (*dict(str, str)*) – Passed to `Request`.

**Keyword Arguments**
****kwargs** – Passed to `send()`.

**Return type**
requests.Response

**caching_get**(*url*, *etag=None*, *last_modified=None*, *headers=None*, ***kwargs*)

Like *get()*, but set and return caching headers.

caching_get(url, etag, last_modified) -> response, etag, last_modified

## Protocols

**class** reader._parser.**FeedArgument**(**args*, ***kwargs*)

Any *FeedForUpdate*-like object.

**property url:** `str`

The feed URL.

**property http_etag:** `str | None`

The HTTP `ETag` header from the last update.

**property http_last_modified:** `str | None`

The the HTTP `Last-Modified` header from the last update.

**class** reader._parser.**RetrieverType**(**args*, ***kwargs*)

A callable that knows how to retrieve a feed.

**slow_to_read:** `bool`

Allow *Parser* to `read()` the result *resource* into a temporary file, and pass that to the parser (as an optimization). Implies the *resource* is a readable binary file.

**__call__**(*url*, *http_etag*, *http_last_modified*, *http_accept*)

Retrieve a feed.

**Parameters**

- **feed** (*str*) – The feed URL.
- **http_etag** (*str or None*) – The HTTP `ETag` header from the last update.
- **http_last_modified** (*str or None*) – The the HTTP `Last-Modified` header from the last update.
- **http_accept** (*str or None*) – Content types to be retrieved, as an HTTP `Accept` header.

**Returns**
A context manager that has as target either the result or `None`, if the feed didn't change.

**Return type**
contextmanager(*RetrieveResult* or None)

**Raises**
*ParseError* –

**validate_url**(*url*)

Check if `url` is valid for this retriever.

**Raises**
*InvalidFeedURLError* – If `url` is not valid.

**class** reader._parser.**FeedForUpdateRetrieverType**(**args*, ***kwargs*)

Bases: *RetrieverType*[T_co], Protocol

A *RetrieverType* that can change update-relevant information.

**process_feed_for_update**(*feed*)

Change update-relevant information about a feed before it is passed to the retriever (*RetrieverType.__call__()*).

**Parameters**
**feed** (FeedForUpdate) – Feed information.

**Returns**
The passed-in feed information, possibly modified.

**Return type**
*FeedForUpdate*

**class** reader._parser.**ParserType**(**args*, ***kwargs*)

A callable that knows how to parse a retrieved feed.

**__call__**(*url*, *resource*, *headers*)

Parse a feed.

**Parameters**

- **resource** (*T_cv*) – The feed resource. Usually, a readable binary file.
- **headers** (*dict(str, str) or None*) – The HTTP response headers associated with the resource.

**Returns**
The feed and entry data.

**Return type**
tuple(*FeedData*, collection(*EntryData*))

**Raises**
*ParseError* –

**class** reader._parser.**HTTPAcceptParserType**(**args*, ***kwargs*)

Bases: *ParserType*[T_cv], Protocol

A *ParserType* that knows what content it can handle.

**property http_accept:** str

The content types this parser supports, as an `Accept` HTTP header value.

**class** reader._parser.**EntryPairsParserType**(**args*, ***kwargs*)

Bases: *ParserType*[T_cv], Protocol

A *ParserType* that can modify entry data before being stored.

**process_entry_pairs**(*url*, *pairs*)

Process entry data before being stored.

**Parameters**

- **url** (*str*) – The feed URL.
- **pairs** (*iterable(tuple(EntryData, EntryForUpdate or None))*) – (entry data, entry for update) pairs.

**Returns**
(entry data, entry for update) pairs, possibly modified.

**Return type**
iterable(tuple(*EntryData*, *EntryForUpdate* or None))

class reader._parser.requests.**RequestHook**(**args*, ***kwargs*)

Hook to modify a `Request` before it is sent.

__call__(*session*, *request*, ***kwargs*)

Modify a request before it is sent.

**Parameters**

- **session** (*requests.Session*) – The session that will send the request.
- **request** (*requests.Request*) – The request to be sent.

**Keyword Arguments**
****kwargs** – Will be passed to `send()`.

**Returns**
A (possibly modified) request to be sent. If none, send the initial request.

**Return type**
requests.Request or None

class reader._parser.requests.**ResponseHook**(**args*, ***kwargs*)

Hook to repeat a request depending on the `Response`.

__call__(*session*, *response*, *request*, ***kwargs*)

Repeat a request depending on the response.

**Parameters**

- **session** (*requests.Session*) – The session that sent the request.
- **request** (*requests.Request*) – The sent request.
- **response** (*requests.Response*) – The received response.

**Keyword Arguments**
****kwargs** – Were passed to `send()`.

**Returns**
A (possibly new) request to be sent, or None, to return the current response.

**Return type**
requests.Request or None

### Data objects

class reader._parser.**RetrieveResult**(*resource*, *mime_type=None*, *http_etag=None*, *http_last_modified=None*, *headers=None*)

The result of retrieving a feed, plus metadata.

**resource:** T_co

The result of retrieving a feed. Usually, a readable binary file. Passed to the parser.

**mime_type:** str | None = None

The MIME type of the resource. Used to select an appropriate parser.

**http_etag:** str | None = None

The HTTP `ETag` header associated with the resource. Passed back to the retriever on the next update.

**http_last_modified:** str | None = None

The HTTP `Last-Modified` header associated with the resource. Passed back to the retriever on the next update.

**headers:** Mapping[str, str] | None = None

The HTTP response headers associated with the resource. Passed to the parser.

class reader._types.**ParsedFeed**(*feed*, *entries*, *http_etag=None*, *http_last_modified=None*, *mime_type=None*)

A parsed feed.

**feed:** *FeedData*

The feed.

**entries:** Iterable[*EntryData*]

Iterable of entries.

**http_etag:** str | None

The HTTP `ETag` header associated with the feed resource. Passed back to the retriever on the next update.

**http_last_modified:** str | None

The HTTP `Last-Modified` header associated with the feed resource. Passed back to the retriever on the next update.

**mime_type:** str | None

The MIME type of the feed resource. Used by *process_entry_pairs()* to select an appropriate parser.

class reader._types.**FeedData**(*url*, *updated=None*, *title=None*, *link=None*, *author=None*, *subtitle=None*, *version=None*)

Feed data that comes from the feed.

Attributes are a subset of those of *Feed*.

**url:** str

**updated:** datetime | None = None

**title:** str | None = None

**link:** str | None = None

**author:** str | None = None

**subtitle:** str | None = None

**version:** str | None = None

**as_feed**(***kwargs*)

Convert this to a feed; kwargs override attributes.

**Returns**
*Feed*.

**Return type**
Feed

**property resource_id:** tuple[str]

**property hash:** bytes

**class** reader._types.**EntryData**(*feed_url*, *id*, *updated=None*, *title=None*, *link=None*, *author=None*, *published=None*, *summary=None*, *content=()*, *enclosures=()*)

Entry data that comes from the feed.

Attributes are a subset of those of *Entry*.

**feed_url:** str

**id:** str

**updated:** datetime | None = None

**title:** str | None = None

**link:** str | None = None

**author:** str | None = None

**published:** datetime | None = None

**summary:** str | None = None

**content:** Sequence[*Content*] = ()

**enclosures:** Sequence[*Enclosure*] = ()

**as_entry**(***kwargs*)

Convert this to an entry; kwargs override attributes.

**Returns**
*Entry*.

**Return type**
Entry

**property resource_id:** tuple[str, str]

**property hash:** bytes

class reader._types.**FeedForUpdate**(*url*, *updated*, *http_etag*, *http_last_modified*, *stale*, *last_updated*, *last_exception*, *hash*)

Update-relevant information about an existing feed, from Storage.

**url:** str
The feed URL.

**updated:** datetime | None
The date the feed was last updated, according to the feed.

**http_etag:** str | None
The HTTP `ETag` header from the last update.

**http_last_modified:** str | None
The HTTP `Last-Modified` header from the last update.

**stale:** bool
Whether the next update should update *all* entries, regardless of their *hash* or *updated*.

**last_updated:** datetime | None
The date the feed was last updated, according to reader; none if never.

**last_exception:** bool
Whether the feed had an exception at the last update.

**hash:** bytes | None
The *hash* of the corresponding FeedData.

class reader._types.**EntryForUpdate**(*updated*, *published*, *hash*, *hash_changed*)

Update-relevant information about an existing entry, from Storage.

**updated:** datetime | None
The date the entry was last updated, according to the entry.

**published:** datetime | None
The date the entry was published, according to the entry.

**hash:** bytes | None
The *hash* of the corresponding EntryData.

**hash_changed:** int | None
The number of updates due to a different `hash` since the last time `updated` changed.

### 4.2.2 Storage

*reader* storage is abstracted by two DAO (data access object) protocols: *StorageType*, which provides the main storage, and *SearchType*, which provides search-related operations.

Currently, there's only one supported implementation, based on SQLite.

That said, it is possible to use an alternate implementation by passing a *StorageType* instance via the `_storage` *make_reader()* argument:

```
reader = make_reader('unused', _storage=MyStorage(...))
```

The protocols are *mostly* stable, but some backwards-incompatible changes are expected in the future (known ones are marked below with *Unstable*). The long term goal is for the storage API to become stable, but at least one other implementation needs to exists before that. (Working on one? *Let me know!*)

**Unstable**

Currently, search is tightly-bound to a storage implementation (see *make_search()*). While the *change tracking API* allows search implementations to keep in sync with text content changes, there is no convenient way for *SearchType.search_entries()* to filter/sort results without storage cooperation; *StorageType* will need additional capabilities to support this.

Reader._storage

The *StorageType* instance used by this reader.

Reader._search

The *SearchType* instance used by this reader.

class reader._types.StorageType

Storage DAO protocol.

For methods with *Reader* correspondents, see the Reader docstrings for detailed semantics.

Any method can raise *StorageError*.

The behaviors described in *Lifecycle* and *Threading* are implemented at the storage level; specifically:

- The storage can be used directly, without *__enter__()*ing it. There is no guarantee *close()* will be called at the end.
- The storage can be reused after *__exit__()* / *close()*.
- The storage can be used from multiple threads, either directly, or as a context manager. Closing the storage in one thread should not close it in another thread.

Schema migrations are transparent to *Reader*. The current storage implementation does them at initialization, but others may require them to happen out-of-band with user intervention.

All `datetime` attributes of all parameters and return values are timezone-aware, with the timezone set to `utc`.

**Unstable**

In the future, implementations will be required to accept datetimes with any timezone.

Methods, grouped by topic:

**object lifecycle**
*__enter__()* *__exit__()* *close()*

**feeds**
*add_feed()* *delete_feed()* *change_feed_url()* *get_feeds()* *get_feed_counts()* *set_feed_user_title()* *set_feed_updates_enabled()*

**entries**
*add_entry()* *delete_entries()* *get_entries()* *get_entry_counts()* *set_entry_read()* *set_entry_important()*

**tags**
*get_tags() set_tag() delete_tag()*

**update**
*get_feeds_for_update() update_feed() set_feed_stale() get_entries_for_update() add_or_update_entries() get_entry_recent_sort() set_entry_recent_sort()*

**__enter__**()
Called when *Reader* is used as a context manager.

**__exit__**(*_)
Called when *Reader* is used as a context manager.

**close**()
Called by *Reader.close()*.

**add_feed**(*url*, */*, *added*)
Called by *Reader.add_feed()*.

**Parameters**
- **url** (*str*) –
- **added** (*datetime*) – *Feed.added*

**Raises**
***FeedExistsError*** –

**delete_feed**(*url*, */*)
Called by *Reader.delete_feed()*.

**Parameters**
**url** (*str*) –

**Raises**
***FeedNotFoundError*** –

**change_feed_url**(*old*, *new*, */*)
Called by *Reader.change_feed_url()*.

**Parameters**
- **old** (*str*) –
- **new** (*str*) –

**Raises**
***FeedNotFoundError*** –

**get_feeds**(*filter*, *sort*, *limit*, *starting_after*)
Called by *Reader.get_feeds()*.

**Parameters**
- **filter** (FeedFilter) –
- **sort** (*Literal['title', 'added']*) –
- **limit** (*int | None*) –

- **starting_after** (*str | None*) –

**Returns**
A lazy iterable.

**Raises**
***FeedNotFoundError*** – If `starting_after` does not exist.

**Return type**
*Iterable*[Feed]

**get_feed_counts**(*filter*)

Called by *Reader.get_feed_counts()*.

**Parameters**
**filter** (FeedFilter) –

**Returns**
The counts.

**Return type**
FeedCounts

**set_feed_user_title**(*url*, *title*, */*)

Called by *Reader.set_feed_user_title()*.

**Parameters**

- **url** (*str*) –
- **title** (*str | None*) –

**Raises**
***FeedNotFoundError*** –

**set_feed_updates_enabled**(*url*, *enabled*, */*)

Called by *Reader.enable_feed_updates()* and *Reader.disable_feed_updates()*.

**Parameters**

- **url** (*str*) –
- **enabled** (*bool*) –

**Raises**
***FeedNotFoundError*** –

**add_entry**(*intent*, */*)

Called by *Reader.add_entry()*.

**Parameters**
**intent** (EntryUpdateIntent) –

**Raises**

- ***EntryExistsError*** –
- ***FeedNotFoundError*** –

**delete_entries**(*entries*, */*, ***, *added_by*)

Called by *Reader.delete_entry()*.

Also called by plugins like *entry_dedupe*.

**Parameters**

- **entries** (*Iterable[tuple[str, str]]*) – A list of *Entry.resource_id*s.
- **added_by** (*str | None*) – If given, only delete the entries if their *added_by* is equal to this.

**Raises**

- ***EntryNotFoundError*** – An entry does not exist.
- ***EntryError*** – An entry added_by is different from the given one.

**get_entries**(*filter*, *sort*, *limit*, *starting_after*)

Called by *Reader.get_entries()*.

**Parameters**

- **filter** (EntryFilter) –
- **sort** (*Literal['recent', 'random']*) –
- **limit** (*int | None*) –
- **starting_after** (*tuple[str, str] | None*) –

**Returns**
A lazy iterable.

**Raises**
***EntryNotFoundError*** – If starting_after does not exist.

**Return type**
*Iterable*[Entry]

**get_entry_counts**(*now*, *filter*)

Called by *Reader.get_entry_counts()*.

---

**Unstable**

In order to expose better feed interaction statistics, this method will need to return more granular data.

---

---

**Unstable**

In order to support *search_entry_counts()* of search implementations that are not bound to a storage, this method will need to take an entries argument.

---

**Parameters**

- **now** (*datetime*) – Time *averages* is relative to.
- **filter** (EntryFilter) –

**Returns**
The counts.

**Return type**
EntryCounts

**set_entry_read**(*entry*, *read*, *modified*, */*)

Called by *Reader.set_entry_read()*.

**Parameters**

- **entry** (*tuple[str, str]*) –
- **read** (*bool*) –
- **modified** (*datetime | None*) –

**Raises**
*EntryNotFoundError* –

**set_entry_important**(*entry*, *important*, *modified*, */*)

Called by *Reader.set_entry_important()*.

**Parameters**

- **entry** (*tuple[str, str]*) –
- **important** (*bool | None*) –
- **modified** (*datetime | None*) –

**Raises**
*EntryNotFoundError* –

**get_tags**(*resource_id*, *key=None*, */*)

Called by *Reader.get_tags()*.

Also called by *Reader.get_tag_keys()*.

**Unstable**

A dedicated `get_tag_keys()` method will be added in the future.

**Unstable**

Both this method and `get_tag_keys()` will allow filtering by prefix (include/exclude), case sensitive and insensitive; implementations should allow for this.

**Parameters**

- **resource_id** (*tuple[()] | tuple[str] | tuple[str, str] | None | tuple[None] | tuple[None, None]*) –
- **key** (*str | None*) –

**Returns**
A lazy iterable.

**Return type**
*Iterable*[tuple[str, reader.types.JSONType]]

**set_tag**(*resource_id: tuple[()] | tuple[str] | tuple[str, str]*, *key: str*, */*) → None

**set_tag**(*resource_id: tuple[()] | tuple[str] | tuple[str, str]*, *key: str*, *value: JSONType*, */*) → None

Called by *Reader.set_tag()*.

**Parameters**

- **resource_id** –
- **key** –

- **value** –

**Raises**
*ResourceNotFoundError* –

**delete_tag**(*resource_id*, *key*, */*)

Called by *Reader.delete_tag()*.

**Parameters**

- **resource_id** (*tuple[()] | tuple[str] | tuple[str, str]*) –
- **key** (*str*) –

**Raises**
*TagNotFoundError* –

**get_feeds_for_update**(*filter*)

Called by update logic.

**Parameters**
**filter** (FeedFilter) –

**Returns**
A lazy iterable.

**Return type**
*Iterable*[FeedForUpdate]

**update_feed**(*intent*, */*)

Called by update logic.

**Parameters**
**intent** (FeedUpdateIntent) –

**Raises**
*FeedNotFoundError* –

**set_feed_stale**(*url*, *stale*, */*)

Used by update logic tests.

**Parameters**

- **url** (*str*) –
- **stale** (*bool*) – *FeedForUpdate.stale*

**Raises**
*FeedNotFoundError* –

**get_entries_for_update**(*entries*, */*)

Called by update logic.

**Parameters**
**entries** (*Iterable[tuple[str, str]]*) –

**Returns**
An iterable of entry or None (if an entry does not exist), matching the order of the input iterable.

**Return type**
*Iterable*[EntryForUpdate | None]

add_or_update_entries(*intents*, /)

Called by update logic.

**Parameters**
**intents** (*Iterable[EntryUpdateIntent]*) –

**Raises**
*FeedNotFoundError* –

get_entry_recent_sort(*entry*, /)

Get *EntryUpdateIntent.recent_sort*.

Used by plugins like *entry_dedupe*.

**Parameters**
**entry** (*tuple[str, str]*) –

**Returns**
entry *recent_sort*

**Raises**
*EntryNotFoundError* –

**Return type**
*datetime*

set_entry_recent_sort(*entry*, *recent_sort*, /)

Set *EntryUpdateIntent.recent_sort*.

Used by plugins like *entry_dedupe*.

**Parameters**

- **entry** (*tuple[str, str]*) –
- **recent_sort** (*datetime*) –

**Raises**
*EntryNotFoundError* –

class reader._types.BoundSearchStorageType

Bases: *StorageType*, Protocol

A storage that can create a storage-bound search provider.

make_search()

Create a search provider.

**Returns**
A search provider.

**Return type**
SearchType

class reader._types.SearchType

Search DAO protocol.

Any method can raise *SearchError*.

There are two sets of methods that may be called at different times:

**management methods**
*enable() disable() is_enabled() update()*

**read-only methods**
*search_entries() search_entry_counts()*

---

**Unstable**

In the future, search may receive object lifecycle methods (context manager + `close()`), to support implementations that do not share state with the storage. If you need support for this, please open a issue.

---

`enable()`

Called by *Reader.enable_search()*.

A no-op and reasonably fast if search is already enabled.

Checks if all dependencies needed for *update()* are available, raises *SearchError* if not.

**Raises**
***StorageError*** –

`disable()`

Called by *Reader.disable_search()*.

`is_enabled()`

Called by *Reader.is_search_enabled()*.

Not called otherwise.

**Returns**
Whether search is enabled or not.

**Return type**
bool

`update()`

Called by *Reader.update_search()*.

Should not enable search automatically (handled by *Reader*).

**Raises**

- ***SearchNotEnabledError*** –
- ***StorageError*** –

`search_entries`(*query*, */*, *filter*, *sort*, *limit*, *starting_after*)

Called by *Reader.search_entries()*.

**Parameters**

- **query** (*str*) –
- **filter** (EntryFilter) –
- **sort** (*Literal['relevant', 'recent', 'random']*) –
- **limit** (*int | None*) –
- **starting_after** (*tuple[str, str] | None*) –

**Returns**
A lazy iterable.

**Raises**

- *SearchNotEnabledError* –
- *InvalidSearchQueryError* –
- *EntryNotFoundError* – If `starting_after` does not exist.

**Return type**
*Iterable*[EntrySearchResult]

**search_entry_counts**(*query*, */*, *now*, *filter*)

Called by *Reader.search_entry_counts()*.

**Parameters**

- **query** (*str*) –
- **now** (*datetime*) – Time *averages* is relative to.
- **filter** (EntryFilter) –

**Returns**
The counts.

**Raises**

- *SearchNotEnabledError* –
- *InvalidSearchQueryError* –
- *StorageError* –

**Return type**
EntrySearchCounts

**class** reader._types.**ChangeTrackingStorageType**

Bases: *StorageType*, Protocol

A storage that can track changes to the text content of resources.

**property changes:** *ChangeTrackerType*

The change tracker associated with this storage.

**class** reader._types.**ChangeTrackerType**

Storage API used to keep the full-text search index in sync.

---

The sync model works as follows.

Each resource to be indexed has a sequence that changes every time its text content changes. The sequence can be a global counter, a random number, or a high-precision timestamp; the only requirement is that it won't be used again (or it's extremely unlikely that will happen).

Each sequence change gets recorded. Updates are recorded as pairs of *DELETE* + *INSERT* changes with the old / new sequences, respectively.

*SearchType.update()* gets changes and processes them. For *INSERT*, the resource is indexed only if the change sequence matches the current main storage sequence; otherwise, the change is ignored. For *DELETE*, the resource is deleted only if the change sequence matches the search index sequence. (This means that, during updates, multiple versions of a resource may appear in the index, with different sequences.) Processed changes are marked as done, regardless of the action taken. Pseudocode:

```
while changes := self.storage.changes.get():
    self._process_changes(changes)
    self.storage.changes.done(changes)
```

Enabling change tracking sets the sequence of all resources and adds matching *INSERT* changes to allow backfilling the search index. The sequence may be `None` when change tracking is disabled. There is no guarantee the sequence of a resource remains the same when change tracking is disabled and then enabled again.

**See also:**

The model was validated using property-based testing in this gist.

---

The entry sequence is exposed as *Entry._sequence*, and should change when the entry *title*, *summary*, or *content* change, or when its feed's *title* or *user_title* change.

As of version 3.12, only entry changes are tracked, but the API supports tracking feeds and tags in the future; search implementations should ignore changes to resources they do not support (but still mark them as done!).

Any method can raise *StorageError*.

**enable**()

Enable change tracking.

A no-op and reasonably fast if change tracking is already enabled.

**disable**()

Disable change tracking.

A no-op if change tracking is already disabled.

**get**(*action=None*, *limit=None*)

Return the next batch of changes, if any.

**Parameters**

- **action** (*Action | None*) – Only return changes of this type.
- **limit** (*int | None*) – Return at most this many changes; may return fewer, depending on storage internal limits. If none, reasonable limit should be used (hundreds).

**Returns**
A batch of changes.

**Raises**
*ChangeTrackingNotEnabledError* –

**Return type**
list[Change]

**done**(*changes*)

Mark changes as done. Ignore unknown changes.

**Parameters**
**changes** (*list[Change]*) –

**Raises**

- *ChangeTrackingNotEnabledError* –
- **ValueError** – If more changes than *get()* returns are passed; `done(get())` should always work.

class reader._types.**Change**(*action*, *sequence*, *resource_id*, *tag_key=None*)

A change to be applied to the search index.

The change can be of an entry, a feed, or a resource tag.

**action:** *Action*

Action to take.

**sequence:** bytes

Resource/tag sequence.

**resource_id:** tuple[()] | tuple[str] | tuple[str, str]

Resource id.

**tag_key:** str | None = None

Tag key, if the change is about a tag.

class reader._types.**Action**(*value*, *names=None*, **values*, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

Action to take.

**INSERT = 1**

The resource needs to be added to the search index.

**DELETE = 2**

The resource needs to be deleted from the search index.

Entry.**_sequence:** bytes | None = None

Change sequence.

May be None when change tracking is disabled.

---

**Unstable**

This field is part of the unstable *change tracking API*.

---

exception reader.exceptions.**ChangeTrackingNotEnabledError**(*message=''*)

Bases: *StorageError*

A change tracking method was called when change tracking was not enabled.

---

**Unstable**

This exception is part of the unstable *change tracking API*.

---

### Data objects

class reader._types.**FeedFilter**(*feed_url=None*, *tags=()*, *broken=None*, *updates_enabled=None*, *new=None*)

Options for filtering the results feed list operations.

See the *Reader.get_feeds()* docstring for detailed semantics.

**feed_url:** str | None
Alias for field number 0

**tags:** TagFilter
Alias for field number 1

**broken:** bool | None
Alias for field number 2

**updates_enabled:** bool | None
Alias for field number 3

**new:** bool | None
Alias for field number 4

class reader._types.**EntryFilter**(*feed_url=None*, *entry_id=None*, *read=None*, *important='any'*, *has_enclosures=None*, *tags=()*, *feed_tags=()*)

Options for filtering the results entry list operations.

See the *Reader.get_entries()* docstring for detailed semantics.

**feed_url:** str | None
Alias for field number 0

**entry_id:** str | None
Alias for field number 1

**read:** bool | None
Alias for field number 2

**important:** TristateFilter
Alias for field number 3

**has_enclosures:** bool | None
Alias for field number 4

**tags:** TagFilter
Alias for field number 5

**feed_tags:** TagFilter
Alias for field number 6

class reader._types.**FeedUpdateIntent**(*url*, *last_updated*, *feed=None*, *http_etag=None*, *http_last_modified=None*, *last_exception=None*)

Data to be passed to Storage when updating a feed.

**url:** str

The feed URL.

**last_updated:** datetime | None

The time at the start of updating this feed.

**feed:** *FeedData* | None

The feed data, if any.

**http_etag:** str | None

The feed's ETag header; see *ParsedFeed.http_etag* for details.

---

**Unstable**

*http_etag* and *http_last_modified* may be grouped in a single attribute in the future.

---

**http_last_modified:** str | None

The feed's Last-Modified header; see *ParsedFeed.http_last_modified* for details.

**last_exception:** *ExceptionInfo* | None

Cause of *UpdateError*, if any; if set, everything else except *url* should be None.

class reader._types.**EntryUpdateIntent**(*entry*, *last_updated*, *first_updated*, *first_updated_epoch*, *recent_sort*, *feed_order=0*, *hash_changed=0*, *added_by='feed'*)

Data to be passed to Storage when updating a feed.

**entry:** *EntryData*

The entry data.

**last_updated:** datetime

The time at the start of updating the feed (start of *update_feed()* in *update_feed()*, start of each feed update in *update_feeds()*).

**first_updated:** datetime | None

First *last_updated* (sets *Entry.added*). None if the entry already exists.

**first_updated_epoch:** datetime | None

The time at the start of updating this batch of feeds (start of *update_feed()* in *update_feed()*, start of *update_feeds()* in *update_feeds()*). None if the entry already exists.

**recent_sort:** datetime | None

Sort key for the *get_entries()* recent sort order.

**feed_order:** int

The index of the entry in the feed (zero-based).

**hash_changed:** int | None

Same as *EntryForUpdate.hash_changed*.

**added_by:** Literal['feed', 'user']

Same as *Entry.added_by*.

**property new:** bool

Whether the entry is new or not.

#### Type aliases

reader._types.**TagFilter**

Like the `tags` argument of *Reader.get_feeds()*, except:

- only the full mutiple-tags-with-disjunction form is used
- tags are represented as *(is negated, tag name)* tuples (the `-` prefix is stripped)

Assuming a `tag_filter_argument()` function that converts *get_feeds()* tags to *TagFilter*:

```
>>> tag_filter_argument(['one'])
[[(False, 'one')]]
>>> tag_filter_argument(['one', 'two'])
[[(False, 'one')], [(False, 'two')]]
>>> tag_filter_argument([['one', 'two']])
[[(False, 'one'), (False, 'two')]]
>>> tag_filter_argument(['one', '-two'])
[[(False, 'one')], [(True, 'two')]]
>>> tag_filter_argument(True)
[[True]]
```

alias of Sequence[Sequence[Union[bool, tuple[bool, str]]]]

reader._types.**TristateFilter**

Like *TristateFilterInput*, but without bool/None aliases.

alias of Literal['istrue', 'isfalse', 'notset', 'nottrue', 'notfalse', 'isset', 'any']

### 4.2.3 Recipes

#### Parsing a feed retrieved with something other than *reader*

Example of using the *reader* internal API to parse a feed retrieved asynchronously with HTTPX:

```
$ python examples/parser_only.py
death and gravity
Has your password been pwned? Or, how I almost failed to search a 37 GB text file in␣
↪under 1 millisecond (in Python)
```

```
import asyncio
import io
import httpx
from reader._parser import default_parser
from werkzeug.http import parse_options_header

url = "https://death.andgravity.com/_feed/index.xml"
meta_parser = default_parser()


async def main():
    async with httpx.AsyncClient() as client:
        response = await client.get(url)
```

(continues on next page)

(continued from previous page)

```
        # to select the parser, we need the MIME type of the response
        content_type = response.headers.get('content-type')
        if content_type:
            mime_type, _ = parse_options_header(content_type)
        else:
            mime_type = None

        # select the parser (raises ParseError if none found)
        parser, _ = meta_parser.get_parser(url, mime_type)

        # wrap the content in a readable binary file
        file = io.BytesIO(response.content)

        # parse the feed; not doing parser(url, file, response.headers) directly
        # because parsing is CPU-intensive and would block the event loop
        feed, entries = await asyncio.to_thread(parser, url, file, response.headers)

        print(feed.title)
        print(entries[0].title)


if __name__ == '__main__':
    asyncio.run(main())
```

CHAPTER

FIVE

# UNSTABLE FEATURES

The following are optional features that are still being worked on. They may become their own packages, get merged into the main library, or be removed in the future.

## 5.1 Command-line interface

*reader* comes with a command-line interface that exposes basic management functionality.

> **Warning:** The CLI is is not fully stable, see the *roadmap* for details.

**Note:** The command-line interface is optional, use the `cli` extra to install its *dependencies*.

Most commands need a database to work. The following are equivalent:

```
python -m reader --db /path/to/db some-command
READER_DB=/path/to/db python -m reader some-command
```

If no database path is given, `~/.config/reader/db.sqlite` is used (at least on Linux).

Add a feed:

```
python -m reader add http://www.example.com/atom.xml
```

Update all feeds:

```
python -m reader update
```

Serve the web application locally (at http://localhost:8080/):

```
python -m reader serve
```

### 5.1.1 Updating feeds

For *reader* to actually be useful as a feed reader, feeds need to get updated and, if full-text search is enabled, the search index needs to be updated.

You can run the `update` command regularly to update feeds (e.g. every hour). Note that *reader* uses the ETag and Last-Modified headers, so, if supported by the the server, feeds will only be downloaded if they changed.

To avoid waiting too much for a new feed to be updated, you can run `update --new-only` more often (e.g. every minute); this will update only newly-added feeds. This is also a good time to update the search index.

You can achieve this using cron:

```
42 * * * *  reader update -v 2>&1 >>"/tmp/$LOGNAME.reader.update.hourly.log"
* * * * *   reader update -v --new-only 2>&1 >>"/tmp/$LOGNAME.reader.update.new.log";␣
↪reader search update 2>&1 >>"/tmp/$LOGNAME.reader.search.update.log"
```

If you are running *reader* on a personal computer, it might also be convenient to run `update` once immediately after boot:

```
@reboot     sleep 60; reader update -v 2>&1 >>"/tmp/$LOGNAME.reader.update.boot.log"
```

### 5.1.2 Reference

#### reader

```
reader [OPTIONS] COMMAND [ARGS]...
```

#### Options

**--db** <db>

Path to the reader database. [default: /home/docs/.config/reader/db.sqlite]

**--plugin** <plugin>

Import path to a reader plug-in. Can be passed multiple times.

**--cli-plugin** <cli_plugin>

Import path to a CLI plug-in. Can be passed multiple times.

**--config** <config>

Path to the reader config.

**Default**
`/home/docs/.config/reader/config.yaml`

**--feed-root** <feed_root>

Directory local feeds are relative to. '' (empty string) means full filesystem access. If not provided, don't open local feeds.

**--version**

Show the version and exit.

### Environment variables

**READER_DB**

Provide a default for *--db*

**READER_PLUGIN**

Provide a default for *--plugin*

**READER_CLI_PLUGIN**

Provide a default for *--cli-plugin*

**READER_CONFIG**

Provide a default for *--config*

### add

Add a new feed.

```
reader add [OPTIONS] URL
```

### Options

**--update, --no-update**

Update the feed after adding it.

**-v, --verbose**

### Arguments

**URL**

Required argument

### config

Do various things related to config.

```
reader config [OPTIONS] COMMAND [ARGS]...
```

### dump

```
reader config dump [OPTIONS]
```

### Options

**--merge, --no-merge**

### list

List feeds or entries.

```
reader list [OPTIONS] COMMAND [ARGS]...
```

### entries

List all the entries.

Outputs one line per entry in the following format:

    <feed URL> <entry link or id>

```
reader list entries [OPTIONS]
```

### feeds

List all the feeds.

```
reader list feeds [OPTIONS]
```

### remove

Remove an existing feed.

```
reader remove [OPTIONS] URL
```

### Options

**-v, --verbose**

### Arguments

**URL**

    Required argument

### search

Do various things related to search.

```
reader search [OPTIONS] COMMAND [ARGS]...
```

### disable

Disable search.

```
reader search disable [OPTIONS]
```

### enable

Enable search.

```
reader search enable [OPTIONS]
```

### entries

Search entries.

Outputs one line per entry in the following format:

    <feed URL> <entry link or id>

```
reader search entries [OPTIONS] QUERY
```

### Arguments

**QUERY**

Required argument

### status

Check search status.

```
reader search status [OPTIONS]
```

### update

Update the search index.

```
reader search update [OPTIONS]
```

#### Options

**-v, --verbose**

### serve

Start a local HTTP reader server.

```
reader serve [OPTIONS]
```

#### Options

**-h, --host** <host>
: The interface to bind to.

**-p, --port** <port>
: The port to bind to.

**--plugin** <plugin>
: Import path to a web app plug-in. Can be passed multiple times.

**-v, --verbose**

#### Environment variables

**READER_APP_PLUGIN**

: Provide a default for *--plugin*

### update

Update one or all feeds.

If URL is not given, update all the feeds.

Verbosity works like this:

: progress bar + final status
-v: + lines
-vv: + warnings
-vvv: + info
-vvvv: + debug

```
reader update [OPTIONS] [URL]
```

#### Options

**--new-only, --no-new-only**

Only update new (never updated before) feeds.

**--workers** <workers>

Number of threads to use when getting the feeds.

**Default**

1

**-v, --verbose**

#### Arguments

**URL**

Optional argument

## 5.2 Web application

*reader* comes with a minimal web application, intended to work across all browsers, including light-weight / text-only ones.

**Warning:** The web application is not fully supported, see the *roadmap* for details.

**Note:** The web application is optional, use the `app` extra to install its *dependencies*.

### 5.2.1 Serving the web application

*reader* exposes a standard WSGI application as `reader._app.wsgi:app`. See the Flask documentation for more details on how to deploy it. The path to the reader database can be configured through the *config file* or the `READER_DB` environment variable.

**Warning:** The web application has no authentication / authorization whatsoever; it is expected a server / middleware will provide that.

An example uWSGI configuration file (probably not idiomatic, from here):

```
[uwsgi]
socket = /apps/reader/uwsgi/sock
manage-script-name = true
mount = /reader=reader._app.wsgi:app
```

(continues on next page)

(continued from previous page)

```
plugin = python3
virtualenv = /apps/reader/
env = READER_CONFIG=/apps/reader/reader.yaml
```

You can also run the web application with the `serve` command. `serve` uses Werkzeug's development server, so it probably won't scale well past a single user.

**Note:** For privacy reasons, you may want to configure your web server to not send a `Referer` header (by setting `Referrer-Policy` header to `same-origin` for all responses; nginx example). The `serve` command does it by default.

If running on a personal computer, you can use cron to run `serve` at boot:

```
@reboot     sleep 60; reader serve -p 8080 2>&1 ) >>"/tmp/$LOGNAME.reader.serve.boot.log"
```

### 5.2.2 Screenshots

#### Main page

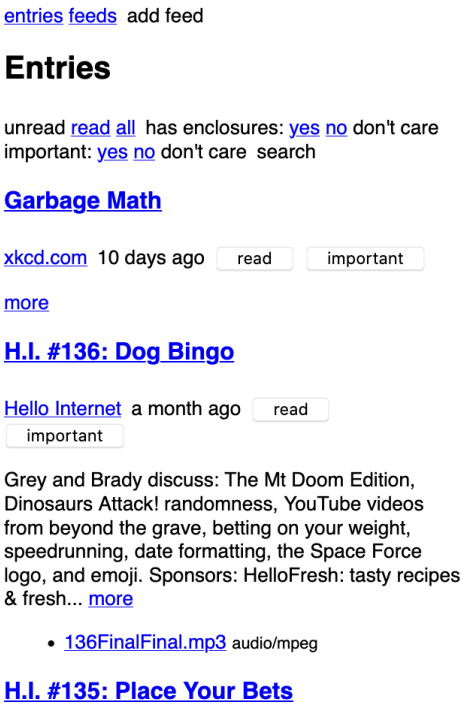


Fig. 1: main page

#### Feed page

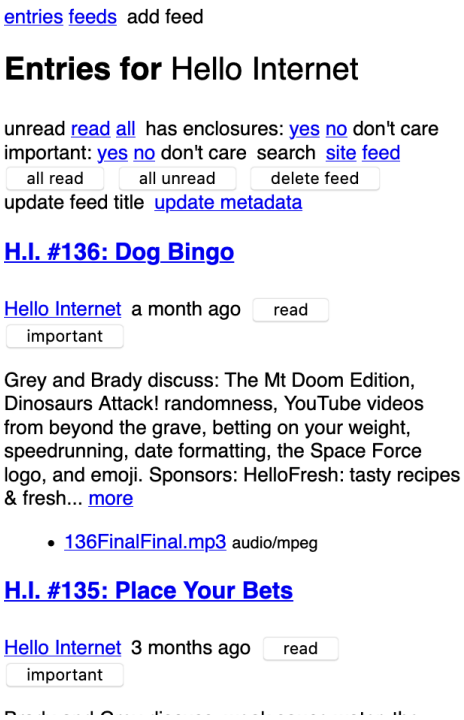


Fig. 2: feed page

#### Feeds page

#### Entry page

#### Search page

#### Lightweight browsers

## 5.3 Configuration

Both the *CLI* and the *web application* can be configured from a file.

> **Warning:** The configuration file format is not stable yet and might change without any notice.

**Note:** Configuration file loading dependencies get installed automatically when installing the CLI or the web application *extras*.

The configuration file path can be specified either through the `--config` CLI option or through the `READER_CONFIG` environment variable (also usable with the web application).

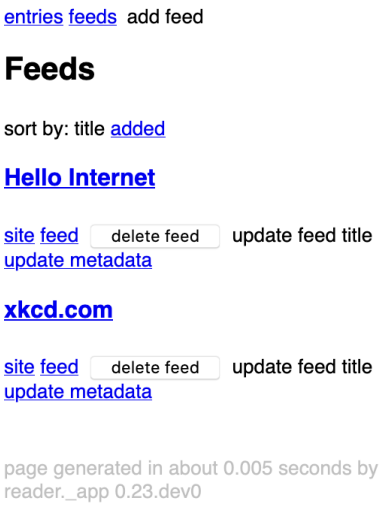


Fig. 3: feeds page

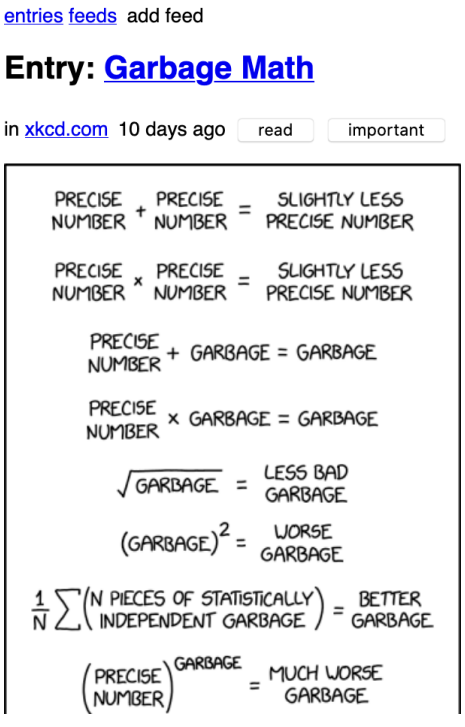


Fig. 4: entry page

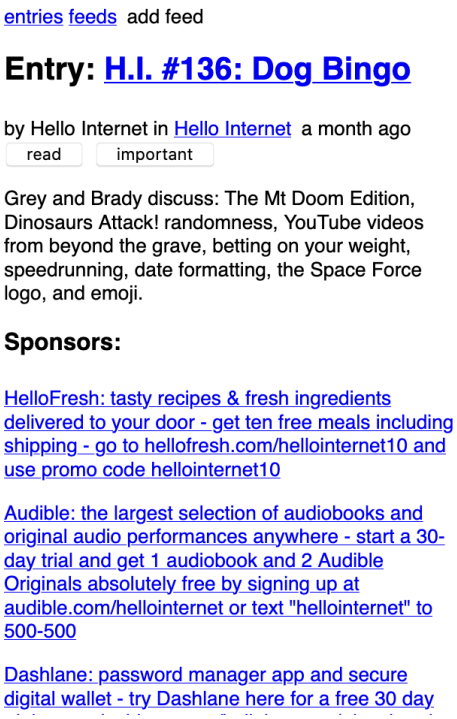


Fig. 5: entry page

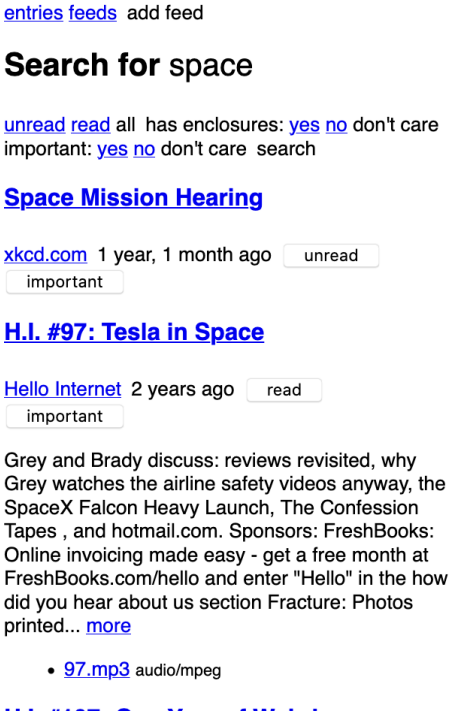


Fig. 6: search page

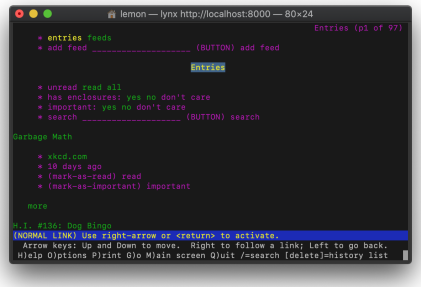


Fig. 7: Lynx

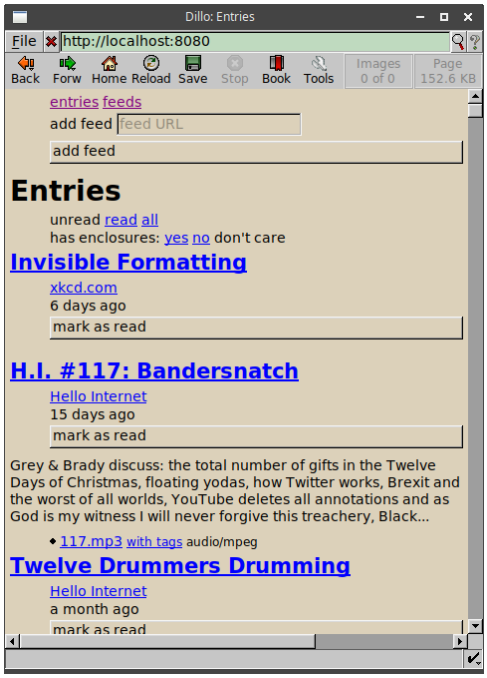


Fig. 8: Dillo

The config file is split in contexts; this allows having a set of global defaults and overriding them with CLI- or web-app-specific values. Use the `config dump --merge` command to see the final configuration for each context.

The older `READER_DB`, `READER_PLUGIN`, and `READER_APP_PLUGIN` environment variables always *replace* the corresponding config values, so they should be used only for debugging.

The following example shows the config file structure and the options currently available:

```
# Contexts are values of the top level map.
# There are 3 known contexts: default, cli, and app.
#
# The default context can also be implicit: top level keys that don't
# correspond to a known context are assumed to belong to the default context.
#
# Thus, the following are equivalent:
#
#   default:
#       reader: ...
#       something else: ...
#
#   ---
#
#   reader: ...
#   something else: ...
#
# However, mixing them is an error:
#
#   default:
#       reader: ...
#   something else: ...


# default context.
#
# Provides default settings for the other contexts.

default:
    # The reader section contains make_reader() keyword arguments:
    reader:
        url: /path/to/db.sqlite
        feed_root: /path/to/feeds

        # Additionally, it's possible to specify reader plugins, as a
        #   <plugin import path>: <plugin options>
        # map; options are ignored at the moment.
        # Note that unlike other settings, plugins are merged, not replaced.
        plugins:
            reader._plugins.sqlite_releases:init:
            reader.ua_fallback:


# CLI context.

cli:
```

(continues on next page)

(continued from previous page)

```
    # When using the CLI, we want to use some additional reader plugins.
    reader:
        plugins:
            reader.mark_as_read:
            reader.entry_dedupe:

    # The cli context also allows changing the CLI defaults.
    defaults:
        # Note that while the --db and --plugin CLI options could appear here,
        # doing it isn't very usful, since the CLI values (including defaults)
        # always override the corresponding config file values.

        # Options that can be passed multiple times take a list of values:
        # --plugin reader._plugins.enclosure_dedupe:enclosure_dedupe
        # plugin: [reader._plugins.enclosure_dedupe:enclosure_dedupe]

        # Subcommand defaults can be given as nested maps:

        # add --update
        add:
            # Flags take a boolean value:
            update: yes

        # update --workers 10 -vv
        update:
            workers: 10
            # Flags that can be repeated take an integer:
            verbose: 2

        search:
            # search update -v
            update:
                verbose: 1

        # serve --port 8888
        serve:
            port: 8888


# Web application context.
#
# Used for both the serve command (`python -m reader serve`)
# and when using the WSGI application (reader._app.wsgi:app) directly.

app:
    # When using the web app, we want to use an additional reader plugin.
    reader:
        plugins:
            reader.enclosure_dedupe:

    # ... and some app plugins.
    plugins:
```

(continues on next page)

(continued from previous page)

```
        reader._plugins.enclosure_tags:init:
        reader._plugins.preview_feed_list:init:
```

## 5.4 Plugins

### 5.4.1 Built-in plugins

This is a list of built-in plugins that are considered stable.

See the *Plugins* section of the user guide for details on how built-in plugins are loaded.

#### reader.enclosure_dedupe

Deduplicate the enclosures of an entry by enclosure URL.

#### reader.entry_dedupe

Deduplicate the entries of a feed.

Sometimes, the format of the entry id changes for all the entries in a feed, for example from `example.com/123` to `example.com/entry-title`. Because *id* uniquely identifies the entries of a feed, this results in them being added again with the new ids.

*entry_dedupe* addresses this by copying entry user attributes like *read* or *important* from the old entries to the new one, and **deleting** the old entries.

Duplicates are entries with the same title *and* the same summary/content.

By default, this plugin runs only for newly-added entries. To run it for the existing entries of a feed, add the `.reader.dedupe.once` tag to the feed; the plugin will run on the next feed update, and remove the tag afterwards. To run it for the existing entries in a feed, and only use the title for comparisons (ignoring the content), use `.reader.dedupe.once.title` instead.

Entry user attributes are set as follows:

*read* / *important*

If any of the entries is read/important, the new entry will be read/important.

*read_modified* / *important_modified*

Set to the oldest *modified* of the entries with the same status as the new read/important.

entry tags

For each tag key:

- collect all the values from the duplicate entries
- if the new entry does not have the tag, set it to the first value
- copy the remaining values to the new entry, using a key of the form `.reader.duplicate.N.of.TAG`, where N is an integer and TAG is the tag key

Only unique values are considered, such that `TAG`, `.reader.duplicate.1.of.TAG`, `.reader.duplicate.2.of.TAG`... always have different values.

To reduce false negatives when detecting duplicates:

- All comparisons are case-insensitive, with HTML tags, HTML entities, punctuation, and whitespace removed.
- For entries with content of different lengths, only a prefix of common (smaller) length is used in comparison. (This is useful when one version of an entry has only the first paragraph of the article, but the other has the whole article.)
- For entries with longer content (over ~48 words), approximate matching is used instead of an exact match (currently, Jaccard similarity of 4-grams).

To reduce false positives when detecting duplicates:

- Titles must match exactly (after clean-up).
- Both entries must have title *and* content.
- Similarity thresholds are set relatively high, and higher for shorter content.

Changed in version 2.3: Delete old duplicates instead of marking them as read / unimportant.

Changed in version 2.2: Reduce false negatives by using approximate content matching.

Changed in version 2.2: Make it possible to re-run the plugin for existing entries.

### reader.mark_as_read

Mark added entries of specific feeds as read + unimportant if their title matches a regex.

To configure, set the `make_reader_reserved_name('mark-as-read')` (by default, `.reader.mark-as-read`) tag to something like:

```
{
    "title": ["first-regex", "second-regex"]
}
```

Changed in version 3.5: Don't set *read_modified* and *important_modified* anymore; because *important* is now optional, `important = False` is enough to mark an entry as unimportant. Old unimportant entries will be migrated automatically.

Changed in version 2.7: Use the `.reader.mark-as-read` metadata for configuration. Feeds using the old metadata, `.reader.mark_as_read`, will be migrated automatically on update until *reader* 3.0.

Changed in version 2.4: Explicitly mark matching entries as unimportant.

### reader.readtime

Calculate the read time for new/updated entries, and store it as the `.reader.readtime` entry tag, with the format:

```
{'seconds': 1234}
```

The content used is that returned by *get_content()*.

The read time for existing entries is backfilled as follows:

- On the first *update_feeds()* / *update_feeds_iter()* call:
  - all feeds with `updates_disabled` false are scheduled to be backfilled
    - the feeds selected to be updated are backfilled then
    - the feeds not selected to be updated will be backfilled the next time they are updated

  - all feeds with `updates_disabled` true are backfilled, regardless of which feeds are selected to be updated

- To prevent any feeds from being backfilled, set the `.reader.readtime` global tag to `{'backfill': 'done'}`.
- To schedule a feed to be backfilled on its next update, set the `.reader.readtime` feed tag to `{'backfill': 'pending'}`.

Changed in version 3.1: Do not require additional dependencies. Deprecate the `readtime` extra.

New in version 2.12.

#### reader.ua_fallback

Retry feed requests that get `403 Forbidden` with a different user agent.

Sometimes, servers blocks requests coming from *reader* based on the user agent. This plugin retries the request with feedparser's user agent, which seems to be more widely accepted.

Servers/CDNs known to not accept the *reader* UA: Cloudflare, WP Engine.

### 5.4.2 Experimental plugins

*reader* also ships with a number of experimental plugins.

For these, the full entry point *must* be specified.

To use them from within Python code, use the entry point as a *custom plugin*:

```
>>> from reader._plugins import sqlite_releases
>>> reader = make_reader("db.sqlite", plugins=[sqlite_releases.init])
```

#### cli_status

Capture the stdout of a CLI command and add it as an entry to a special feed.

The feed URL is `reader:status`; if it does not exist, it is created.

The entry id is the command, without options or arguments:

```
('reader:status', 'command: update')
('reader:status', 'command: search update')
```

Entries are marked as read.

To load:

```
READER_CLI_PLUGIN='reader._plugins.cli_status.init_cli' \
python -m reader ...
```

### preview_feed_list

If the feed to be previewed is not actually a feed, show a list of feeds linked from that URL (if any).

This plugin needs additional dependencies, use the `unstable-plugins` extra to install them:

```
pip install reader[unstable-plugins]
```

To load:

```
READER_APP_PLUGIN='reader._plugins.preview_feed_list:init' \
python -m reader serve
```

Implemented for https://github.com/lemon24/reader/issues/150.

### enclosure_tags

Fix tags for MP3 enclosures (e.g. podcasts).

Adds a "with tags" link to a version of the file with tags set as follows:

- the entry title as title
- the feed title as album
- the entry/feed author as author

This plugin needs additional dependencies, use the `unstable-plugins` extra to install them:

```
pip install reader[unstable-plugins]
```

To load:

```
READER_APP_PLUGIN='reader._plugins.enclosure_tags:init' \
python -m reader serve
```

Implemented for https://github.com/lemon24/reader/issues/50. Became a plugin in https://github.com/lemon24/reader/issues/52.

### sqlite_releases

Create a feed out of the SQLite release history pages at:

- https://www.sqlite.org/changes.html
- https://www.sqlite.org/chronology.html

Also serves as an example of how to write custom parsers.

This plugin needs additional dependencies, use the `unstable-plugins` extra to install them:

```
pip install reader[unstable-plugins]
```

To load:

```
READER_PLUGIN='reader._plugins.sqlite_releases:init' \
python -m reader ...
```

#### timer

Measure *Reader*, Storage, and search method calls, including time spent in iterables.

If loaded, the *Web application* will show per-request method statistics in the footer.

Once `reader.timer.enable()` is called, the timing of each method call is collected in `reader.timer.calls`; `disable()` clears the list of calls and stops collection:

```
>>> reader = make_reader('db.sqlite', plugins=[
...     'reader._plugins.timer:init_reader'
... ])
>>> reader.timer.enable()
>>> for _ in reader.get_entries(limit=500): pass
>>> for call in reader.timer.calls:
...     print(f"{call.name:30} {call.time:9.6f}")
...
Reader.get_entries              0.304127
Storage.get_entries             0.290139
Storage.get_entries_page        0.159803
Storage.get_db                  0.000008
Storage.get_entries_page        0.128641
Storage.get_db                  0.000826
>>> print(reader.timer.format_stats())
                           len    sum    min    avg    max
Reader.get_entries           1  0.304  0.304  0.304  0.304
Storage.get_db               2  0.001  0.000  0.000  0.001
Storage.get_entries          1  0.290  0.290  0.290  0.290
Storage.get_entries_page     2  0.288  0.129  0.144  0.160
```

This plugin needs additional dependencies, use the `unstable-plugins` extra to install them:

```
pip install reader[unstable-plugins]
```

#### share

Add social sharing links at the end of the entry page.

To load:

```
READER_APP_PLUGIN='reader._plugins.share:init' \
python -m reader serve
```

### 5.4.3 Discontinued plugins

Following are experimental plugins that are not very useful anymore.

#### twitter

Prior to version 3.7, *reader* had a Twitter plugin; it was removed because it's not possible to get tweets using the free API tier anymore.

However, the plugin used the internal *Parser* API *in new and interesting ways* – it mapped the multiple tweets in a thread to a single entry, and stored old tweets alongside the rendered HTML content to avoid retrieving them again when updating the thread/entry.

You can still find the code on GitHub: twitter.py.

#### tumblr_gdpr

Prior to version 3.7, *reader* had a plugin to accept Tumblr GDPR terms (between 2018 and 2020, Tumblr would redirect all new sessions to an "accept the terms of service" page, including machine-readable RSS feeds).

This plugin is a good example of how to set cookies on the Requests session used to retrieve feeds.

You can still find the code on GitHub: tumblr_gdpr.py.

### 5.4.4 Loading plugins from the CLI and the web application

There is experimental support of plugins in the CLI and the web application.

> **Warning:** The plugin system/hooks are not stable yet and may change without any notice.

To load plugins, set the `READER_PLUGIN` environment variable to the plugin entry point (e.g. `package.module:entry_point`); multiple entry points should be separated by one space:

```
READER_PLUGIN='first.plugin:entry_point second_plugin:main' \
python -m reader some-command
```

For *built-in plugins*, it is enough to use the plugin name (`reader.XYZ`).

> **Note:** `make_reader()` ignores the plugin environment variables.

To load web application plugins, set the `READER_APP_PLUGIN` environment variable. To load CLI plugins (that customize the CLI), set the `READER_CLI_PLUGIN` environment variable.

CHAPTER

SIX

# PROJECT INFORMATION

*reader* is released under the BSD license, its documentation lives at Read the Docs, the code on GitHub, and the latest release on PyPI. It is rigorously tested on Python 3.10+ and PyPy.

## 6.1 How to contribute to *reader*

Thank you for considering contributing to *reader*!

### 6.1.1 Reporting issues

Please report issues via GitHub Issues.

Include the following information:

- Describe what you expected to happen.
- If possible, include a minimal reproducible example to help identify the issue. This also helps check that the issue is not with your own code.
- Describe what actually happened. Include the full traceback if there was an exception.
- List your Python and *reader* versions. If possible, check if this issue is already fixed in the latest release or the latest code in the repository.

### 6.1.2 Questions

Please use Github Discussions for support or general questions.

### 6.1.3 Submitting patches

If there is no open issue for what you want to submit, prefer opening one for discussion before working on a pull request.

You can work on any help wanted issue that does not have an open PR or an assigned maintainer (no need to ask).

For other open issues, please ask first, there may be background that didn't end up in the issue yet; also see *Roadmap* and *Design notes*.

Include the following in your patch:

- Use Black to format your code. This and other tools will run automatically if you install pre-commit using the instructions below.
- Include tests if your patch adds or changes code. Make sure the test fails without your patch.

- Update any relevant documentation pages and docstrings; also see *Documentation*. Documentation pages and docstrings should be wrapped at 72 characters.
- Add an entry in `CHANGES.rst`. Use the same style as other entries. Also include `.. versionchanged::` inline changelogs in relevant docstrings.

### First time setup

- Make sure you have a GitHub account.
- Download and install the latest version of git.
- Configure git with your username and email.

```
$ git config --global user.name 'your name'
$ git config --global user.email 'your email'
```

- Fork *reader* to your GitHub account by clicking the Fork button.
- Clone your fork locally, replacing `your-username` in the command below with your actual username.

```
$ git clone https://github.com/your-username/reader
$ cd reader
```

- Create a virtualenv. Use the latest version of Python.
  - Linux/macOS

```
$ python3 -m venv .venv
$ . .venv/bin/activate
```

  - Windows

```
> py -3 -m venv .venv
> .venv\Scripts\activate
```

- Install *reader* in editable mode, with development dependencies.

```
$ pip install -e '.[dev]'
```

- Install the pre-commit hooks.

```
$ pre-commit install --install-hooks
```

- Alternatively, use *run.sh* to do the last two steps.

```
$ ./run.sh install-dev
```

### Start coding

- Create a branch to identify the issue you would like to work on. Branch off of the "master" branch.

```
$ git fetch origin
$ git checkout -b your-branch-name origin/master
```

- Using your favorite editor, make your changes, committing as you go.
- Include tests that cover any code changes you make. Make sure the test fails without your patch. Run the tests as described below.
- Push your commits to your fork on GitHub and create a pull request. Link to the issue being addressed with `fixes #123` in the pull request description.

```
$ git push --set-upstream origin your-branch-name
```

### Running the tests

Run the basic test suite with pytest.

```
$ pytest --runslow
```

This runs the tests for the current environment, which is usually sufficient. CI will run the full suite when you submit your pull request. You can run the full test suite with tox if you don't want to wait.

```
$ tox
```

### Running test coverage

Generating a report of lines that do not have test coverage can indicate what code needs to be tested. Use *run.sh* to run `pytest` using `coverage`, generate a report, and check required coverage.

```
$ ./run.sh coverage-all
```

Open `htmlcov/index.html` in your browser to explore the report.

The library **must** have 100% test coverage; the unstable plugins, CLI, and web app do not have coverage requirements.

Read more about coverage.

### Type checking

Run type checking with `mypy`.

```
$ mypy --strict src
```

The library **must** pass strict type checking; the plugins, CLI, and web app do not have type checking requirements.

Read more about mypy.

### Building the docs

Build the docs using Sphinx.

```
$ make -C docs html
```

Open `docs/_build/html/index.html` in your browser to view the docs.

Read more about Sphinx.

### run.sh

```
$ ./run.sh command [argument ...]
```

The run.sh script wraps the steps above as "executable documentation".

**`./run.sh install-dev`**
: *First time setup* (install *reader* and pre-commit hooks)

**`./run.sh test` / `./run.sh test-all`**
: *Running the tests*

**`./run.sh coverage-all`**
: *Running test coverage*

**`./run.sh typing`**
: *Type checking*

**`./run.sh docs`**
: *Building the docs*

Arguments are usually passed along to the underlying tool, e.g. `typing` arguments are passed to `pytest`; see the script source for details.

If you have entr installed, `test-dev`, `typing-dev`, and `docs-dev` will run the corresponding commands when the files in the repo change.

Likewise, `serve-dev` will run the web app with the Flask development server.

## 6.2 Development

Development should follow a problem-solution approach.

**See also:**

*The reader philosophy*

### 6.2.1 Roadmap

The plan is to continue evolving the library to support as many "feed reader application" use cases as possible, while still following the *The reader philosophy*. Even if a specific feature is not a good fit for the library itself, it should be possible to find a more generic solution that makes it possible to build the feature on top.

Following is an unsorted, non-exhaustive list of known areas for improvement. I am working on *reader* based on my current interests, in my spare time, but I will prioritize supporting *contributors* (discussions, reviews and so on).

- OPML support, #165
- *deleting entries*
- *feed interaction statistics*
- security
  - XML safety, #212
  - sanitization unification, likely as a plugin, #125 and #227
  - relative link resolution unification, #125
- sorting
  - reverse order, #201
  - sort by "recently interacted with", #294
  - better feed title sort, #250
  - sort feeds by entry counts
    - by unread entries, #245
    - by `averages` (implemented in the web app, but not in core)
    - *sorting by tag values* can help do this in a plugin
- resource tags
  - *searchable tag values*, e.g. for comments
  - *unification with entry.read/important*
  - optimistic locking, #308
  - filter tags by prefix, #309
- HTTP compliance, likely as plugins
  - 301 Moved Permanently, #246
  - 410 Gone, #246
  - 429 Too Many Requests, #307
- add more fields to data objects
  - entry source, #276
  - extra data, as an escape hatch, #277
- *multiple storage implementations*
- *batch get methods*
- *Internal API* stabilization
- arbitrary website scraping, #222

- *feed categories*, likely as a plugin

**See also:**

Open issues and *Design notes*.

#### Command-line interface

The *Command-line interface* is more or less stable,*[0] although both the output and config loading need more polish and additional tests.

A full-blown terminal feed reader is *not* in scope, since I don't need one, but I'm not opposed to the idea.

#### Web application

The *Web application* is "unsupported", in that it's not all that polished, and I don't have time to do major improvments. But, I am using it daily, and it will keep working until a better one exists.

Long term, I'd like to:

- re-design it from scratch to improve usability (see #318 for a wishlist)
- switch to htmx instead of using a home-grown solution
- spin it off into a separate package/project

### 6.2.2 Backwards compatibility

*reader* uses semantic versioning.

Breaking compatibility is done by incrementing the major version, announcing it in the *Changelog*, and raising deprecation warnings for at least one minor version before the new major version is released (if possible).

There may be minor exceptions to this, e.g. bug fixes and gross violation of specifications; they will be announced in the *Changelog* with a **This is a minor compatibility break** warning.

Schema migrations for the default storage must happen automatically. Migrations can be removed in new major versions, with at least 3 months provided since the last migration.

#### What is the public API

*reader* follows the PEP 8 definition of public interface.

The following are part of the public API:

- Every interface documented in the *API reference*.
- Any (documented) module, function, object, method, and attribute, defined in the *reader* package, that is accessible without passing through a name that starts with underscore.
- The number and position of positional arguments.
- The names of keyword arguments.
- Argument types (argument types cannot become more strict).
- Attribute types (attribute types cannot become less strict).

[0] With the exception of `serve`, which is provided by the web app.

Undocumented type aliases (even if not private) are **not** part of the public API.

Other exceptions are possible; they will be marked aggresively as such.

**See also:**

The Twisted Compatibility Policy, which served as inspiration for this.

#### Internal API

The *Internal API* is not stable, but the long term goal is for it to become so.

In order to support / encourage potential users (e.g. plugins, alternate storage implementations), changes should at least be announced in the *Changelog*.

#### Supported Python versions

The oldest Python version reader should support is:

- the newest CPython available on the latest Ubuntu LTS (3 months after LTS release)
- at least 1 stable PyPy version

This usually ends up being the last 3 stable CPython versions.

Dropping support for a Python version should be announced at least 1 release prior.

### 6.2.3 Releases

For convenience, *reader* only releases major and minor versions (bugfixes go in minor versions). Changes go only to the next release (no backports).

#### Making a release

**Note:** scripts/release.py already does most of these.

Making a release (from `x` to `y` == `x + 1`):

- (release.py) bump version in `src/reader/__init__.py` to `y`
- (release.py) update changelog with release version and date
- (release.py) make sure tests pass / docs build
- (release.py) clean up dist/: `rm -rf dist/`
- (release.py) build tarball and wheel: `python -m build`
- (release.py) push to GitHub
- (release.py prompts) wait for GitHub Actions / Codecov / Read the Docs builds to pass
- upload to test PyPI and check: `twine upload --repository-url https://test.pypi.org/legacy/ dist/*`
- (release.py) upload to PyPI: `twine upload dist/*`
- (release.py) tag current commit with <*major*>.<*minor*> and <*major*>.*x* (e.g. when releasing *1.20*: *1.20* and *1.x*)

- (release.py prompts) create release in GitHub
- build docs from latest and enable `y` docs version (should happen automatically after the first time)
- (release.py) bump versions from `y` to `(y + 1).dev0`, add `(y + 1)` changelog section
- (release.py prompts) trigger Read the Docs build for <*major*>.*x* (doesn't happen automatically)

### 6.2.4 Documentation

Following are notes about what documentation should look like, especially for the stable high-level *API*, since that's what most users will see.

We prefer type information in the method description, not in the signature, since the result is more readable. For the same reason, we prefer hand-written Sphinx-style field list types.

We still use autodoc-provided type hints as fallback for parameters that don't have hand-written types, for type documentation for dataclasses, and for the unstable *Internal API*, where it's too much effort to maintain hand-written types.

Known issues (October 2023, Sphinx version ~7):

- Overloads are shown with full annotation regardless of autodoc_typehints (known, documented behavior). May get better with https://github.com/sphinx-doc/sphinx/issues/10359.
- Type aliases that do not come from hand-written types but from the autodoc typehints are expanded in-place; this also affects the overload type annotations. The documented work-around is to add the aliases to autodoc_type_aliases.
- Type alias names that appear in parameter types do not link to the documentation in *Type aliases*. May get better with https://github.com/sphinx-doc/sphinx/issues/9705

### 6.2.5 Design notes

Folowing are various design notes that aren't captured somewhere else (either in the code, or in the issue where a feature was initially developed).

#### Why use SQLite and not SQLAlchemy?

tl;dr: For "historical reasons".

In the beginning:

- I wanted to keep things as simple as possible, so I don't get demotivated and stop working on it. I also wanted to try out a "problem-solution" approach.
- I think by that time I was already a great SQLite fan, and knew that because of the relatively single-user nature of the thing I won't have to change databases because of concurrency issues.
- The fact that I didn't know exactly where and how I would deploy the web app (and that SQLite is in stdlib) kinda cemented that assumption.

Since then, I did come up with some of my own complexity: there's a SQL query builder, a schema migration system, and there were *some* concurrency issues. SQLAlchemy would have likely helped with the first two, but not with the last one (not without dropping SQLite).

Note that it is possible to use a different storage implementation; all storage stuff happens through a DAO-style interface, and SQLAlchemy was the main real alternative I had in mind. The API is private at the moment (1.10), but if anyone wants to use it I can make it public.

It is unlikely I'll write a SQLAlchemy storage myself, since I don't need it (yet), and I think testing it with multiple databases would take quite some time.

### Multiple storage implementations

Detailed requirements and API discussion: #168#issuecomment-642002049.

Minimal work needed to support alternate storages: #168#issuecomment-1383127564.

Storage internal API documented in version 3.10 (November 2023) in #325.

### Database optimization

Optimization lessons learned while fixing "database is locked" errors: #175#issuecomment-657495233.

Some general guidance on schema/index design: #327#issuecomment-1859147186.

Speeding up `get_entries(sort='recent')`:

- first attempt at adding indexes: #134
- using a computed column (`recent_sort`) didn't change things very much: #279
- an index on `recent_sort` alone is not enough for pagination, the index needs to match 1:1 the WHERE clause: #330.

Speeding up `get_entry_counts(feed=...)`:

- having an index on entries(feed) yielded a 4x improvement: #251
- even better, we should cache commonly-used counts: #306#issuecomment-1694655504

### Parser

file:// handling, feed root, per-URL-prefix parsers (later retrievers, see below):

- requirements: #155#issuecomment-667970956
- detailed requirements: #155#issuecomment-672324186
- method for URL validation: #155#issuecomment-673694472, #155#issuecomment-946591071

Requests session plugins:

- requirements: #155#issuecomment-667970956
- why the Session wrapper exists: #155#issuecomment-668716387 and #155#issuecomment-669164351

Retriever / parser split:

- #205#issuecomment-766321855
- split exception hierarchy (not implemented as of 3.9): #218#issuecomment-1687094315

Alternative feed parsers:

- the logical pipeline of parsing a feed: #264#issuecomment-973190028
- comparison between feedparser and Atoma: #264#issuecomment-981678120, #263

Lessons learned from the *twitter* plugin:

- It is useful for a retriever to pass an arbitrary resource to the parser.

  This is already codified in *RetrieverType()* and *ParserType()* being generic.

- It is useful for a Retriever to store arbitrary caching data; the plugin (mis)used *http_etag* to store the (integer) id of the newest tweet in the thread.

  It would be nice to formalize this into a single "arbitrary caching data" attribute; also see this comment.

- It is useful for a Retriever to pass arbitrary data to itself; the plugin (mis)used *http_etag* to pass from *process_feed_for_update()* to *__call__()*:

  - the bearer token and the ids of recent entries (used to retrieve tweets)
  - the ids of entries to re-render, triggered by a one-off tag (passed along to the parser)

  This distinction was made so that `process_feed_for_update()` takes all the decisions upfront (possibly taking advantage of `Storage.get_feeds_for_update()` future optimisations to e.g. also get tags), and calling the retriever (in parallel) doesn't do any reader operations.

  It would be nice to formalize this as well.

- A plugin can coordinate between a custom retriever and custom parser with an unregistered RetrieveResult MIME type (e.g. `application/x.twitter`).
- A plugin can keep arbitrary data as a content with an unregistered type (e.g. `application/x.twitter+json`).

### Metrics

Some thoughts on implementing metrics: #68#issuecomment-450025175.

Per-call timings introduced in the *timer* experimental plugin.

### Query builder

Survey of possible options: #123#issuecomment-582307504.

In 2021, I've written an entire series about it: https://death.andgravity.com/query-builder

### Pagination for methods that return iterators

Why do it for the private implementation: #167#issuecomment-626753299;

Detailed requirements and API discussion for public pagination: #196#issuecomment-706038363.

### Search

From the initial issue:

- detailed requirements and API discussion: #122#issuecomment-591302580
- discussion of possible backend-independent search queries: #122#issuecomment-508938311

Enabling search by default, and alternative search APIs: #252.

Change tracking API: #323#issuecomment-1930756417, model validated in this gist.

External resources:

- Comprehensive, although a bit old (2017): What every software engineer should know about search (full version)

### reader types to Atom mapping

This whole issue: #153.

### Sort by random

Some thoughts in the initial issue: #105.

### Sort by tag values

It may be useful to be able to sort by tag values in order to allow sorting by cached entry counts: #306#issuecomment-1694655504.

### Entry/feed "primary key" attribute naming

This whole issue: #159#issuecomment-612914956.

### Change feed URL

From the initial issue:

- use cases: #149#issuecomment-700066794
- initial requirements: #149#issuecomment-700532183

### Resource tags / metadata

### Feed tags

Detailed requirements and API discussion, and a case study of how to implement categories on top of tags: #184#issuecomment-689587006.

Merging tags and metadata, and the addition of a new, generic (global, feed, entry) tag API: #266#issuecomment-1013739526.

### Entry tags

#228#issuecomment-810098748 discusses three different kinds of entry user data, how they would be implemented, and why I want more use-cases before implementing them (basically, YAGNI):

- entry searchable text fields (for notes etc.)
- entry tags (similar to feed tags, may be used as additional bool flags)
- entry metadata (similar to feed metadata)
  - also discusses how to build an enclosure cache/preloader (doesn't need special *reader* features besides what's available in 1.16)

#253 discusses using entry tags to implement the current entry flags (read, important); tl;dr: it's not worth adding entry tags just for this. #327 discusses using entry tags for `has_enclosures`; tl;dr: it wouldn't save a lot of code, it would be only *a bit* slower, and it reconfirms that read and important are integral to the data model, so we still want them as regular columns.

After closing #228 with *wontfix* in late 2021, in early 2022 (following the #266 tag/metadata unification) I implemented entry and global tags in #272; there's a list of known use cases in the issue description.

### Resource tags

Optimistic locking for tags: #308.

Filter tags by prefix: #309.

### User-added entries

Discussion about API/typing, and things we didn't do: #239.

### Feed updates

Some thoughts about adding a `map` argument: #152#issuecomment-606636200.

How `update_feeds()` is like a pipeline: comment.

Data flow diagram for the update process, as of v1.13: #204#issuecomment-779709824.

`update_feeds_iter()`:

- use case: #204#issuecomment-779893386 and #204#issuecomment-780541740
- return type: #204#issuecomment-780553373

Disabling updates:

- #187#issuecomment-706539658
- #187#issuecomment-706593497

Updating entries based on a hash of their content (regardless of `updated`):

- stable hasing of Python data objects: #179#issuecomment-796868555, the `reader._hash_utils` module, death and gravity article
- ideas for how to deal with spurious hash changes: #225

Decision to ignore feed.updated when updating feeds: #231.

### Deleting entries

Requirements, open questions, and how it interacts with *`entry_dedupe`*: #96.

A summary of why it isn't easy to do: #301#issuecomment-1442423151.

### Counts API

Detailed requirements and API discussion: #185#issuecomment-731743327.

Tracking additional statistics (e.g. `read_modified`): #254; how to expose said statistics: #254#issuecomment-1807064610.

Notebook with a successful attempt to determine a feed "usefulness" score based on how many entries I mark as read / important / don't care; highlights a number of gaps in the *reader* API: https://gist.github.com/lemon24/93222ef4bc4a775092b56546a6e6cd0f

### Using None as a special argument value

This comment: #177#issuecomment-674786498.

### Batch methods

Some initial thoughts on batch get methods (including API/typing) in #191 (closed with *wontfix*, for now).

Why I want to postpone batch update/set methods: #187#issuecomment-700740251.

tl:dr: Performance is likely a non-issue with SQLite, convenience can be added on top as a plugin.

See the 2.12 reader._app.ResourceTags class for an idea of how to represent a bunch of tags in a reserved-name-scheme-agnostic way (useful e.g. for when get_entries() should return tags x, y, z of each entry).

Some web app measurements that show a few cases where batch methods may help: #306#issuecomment-1694655504.

### Using a single Reader objects from multiple threads

Some thoughts on why it's difficult to do: #206#issuecomment-751383418.

Requirements and use cases: #206#issuecomment-1179739301.

When/how to run `pragma optimize`: #206#issuecomment-1183660880.

Full support added in version 2.16 (July 2022).

### Plugins

List of potential hooks (from mid-2018): #80.

Minimal plugin API (from 2021) – case study and built-in plugin naming scheme: #229#issuecomment-803870781.

We'll add / document new (public) hooks as needed.

"Tag before, clear tag after" pattern for resilient plugins: #246#issuecomment-1596097300.

Update hook error handling:

- expected behavior: #218#issuecomment-1595691410, #218#issuecomment-1666823222
- update hook pseudocode + exception hierarchy: #218#issuecomment-1666869215

### Reserved names

Requirements, thoughts about the naming scheme and prefixes unlikely to collide with user names: #186 (multiple comments).

### Wrapping underlying storage exceptions

Which exception to wrap, and which not: #21#issuecomment-365442439.

In version 3.10 (November 2023), all internal APIs were changed to use timezone-aware datetimes, with the timezone set to UTC, in preparation for support for any timezone.

### Timezone handling

Aware vs. naive, and what's needed to go fully aware: #233#issuecomment-881618002.

### OPML support

Thoughts on dynamic lists of feeds: #165#issuecomment-905893420.

### entry_dedupe

Using MinHash to speed up similarity checks (maybe): https://gist.github.com/lemon24/b9af5ade919713406bda9603847d32e5

### REST API

Some early thoughts: #192#issuecomment-700773138 (closed with *wontfix*, for now).

### Web application

### Web interface design philosophy

The web interface should be as minimal as possible.

The web interface should work with text-only browsers, modern browsers, and everything in-between. Some may be nicer to use, but all functionality should be available everywhere.

Fast and ugly is better than slow and pretty.

It should be possible to build a decent web interface (at least for reader) using only HTML forms with a few JavaScript enhancements added on top.

2023 update: Hypermedia Systems and htmx seem to embody these ideas in a much better way than I could; a potential web app re-design will likely use them.

### User interactions

**Note:** This list might lag behind reality; anyway, it all started from here.

User interactions, by logical groups:

- entry
  - mark an entry as read
  - mark an entry as unread
  - go to an entry's link
  - go to an entry's feed
  - go to an entry's feed link
- entry list
  - see the latest unread entries
  - see the latest read entries
  - see the latest entries
- entry list (feed)
  - mark all the entries as read
  - mark all the entries as unread
- feed
  - add a feed
  - delete a feed
  - change a feed's title
  - go to a feed's entries
  - go to a feed's link
- feed list
  - see a list of all the feeds
- other
  - be notified of the success/failure of a previous action

Controls (below), mapped to user interactions:

- link
  - go to …
  - see …
- simple button
  - mark an entry as read
  - mark an entry as unread
- button with input

  - add a feed
  - change a feed's title
- button with checkbox
  - mark all the entries are read
  - mark all the entries are unread
  - delete a feed

#### Controls

There are three interaction modes, HTML-only, HTML+CSS, and HTML+CSS+JS. Each mode adds enhancements on top of the previous one.

In the HTML-only mode, all elements of a control are visible. Clicking the element that triggers the action (e.g. a button) submits a form and, if possible, redirects back to the source page, with any error messages shown after the action element.

In the HTML+CSS mode, some elements might be hidden so that only the action element is visible; in its inert state it should look like text. On hover, the other elements of the control should become visible.

In the HTML+CSS+JS mode, clicking the action element results in an asynchronous call, with the status of the action displayed after it.

Links are just links.

Simple buttons consist of a single button.

Buttons with input consist of an text input element followed by a button. The text input are hidden when not hovered.

Buttons with checkbox consist of a checkbox, a label for the checkbox, and a button. The checkbox and label are hidden when not hovered.

#### Page structure

Text TBD.

#### Pages

Text TBD.

## 6.3 Changelog

### 6.3.1 Version 3.12

Released 2024-03-05

- Split the *full-text search* index into a separate, attached database. (#323)
- Require at least SQLite 3.18. Previously, *reader* core required 3.15, and only `update_search()` required 3.18. (#323)
- Enable write-ahead logging only once, when the database is created, instead of every time it is opened. (#323)

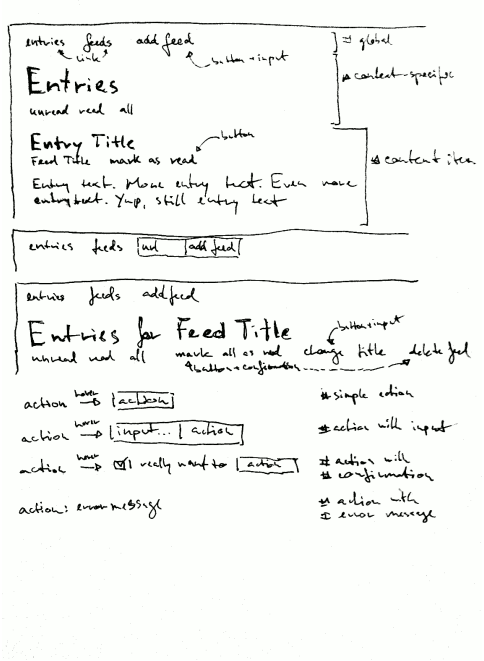


Fig. 1: page structure, controls

- Vacuum the main database after migrations. (#323)
- Add an internal *change tracking API* to formalize how search keeps in sync with storage. (#323)
- Refactor storage internals. (#323)

### 6.3.2 Version 3.11

Released 2023-12-30

- Allow filtering entries by their (entry) tags. (#328)
- Support Python 3.12. (#326)

### 6.3.3 Version 3.10

Released 2023-11-12

- Stop using deprecated `sqlite3` datetime converters/adapters. (#321)
- Document the storage *Internal API*. (#325)
- Change all *internal APIs* to use timezone-aware datetimes, with the timezone set to UTC. (#321)
- In the API documentation, fall back to type hints if hand-written parameter types are not available. Add relevant *Documentation* guidelines to the dev documentation. (#287)
- Add the `share` experimental plugin to add social sharing links in the web app.

### 6.3.4 Version 3.9

Released 2023-08-28

- Wrap unexpected retriever/parser errors in *ParseError*, instead of letting them bubble up, so exceptions raised by custom retrievers/parsers for one feed don't prevent updates for the others during *update_feeds_iter()* / *update_feeds()*. (#218)
- Store the details of any *UpdateError* in *Feed.last_exception* (except hook errors), not just the `__cause__` of *ParseError*s. (#218)
- Add the *timer* experimental plugin to collect per-call method timings. Show per-request statistics in the web app. (#306)

### 6.3.5 Version 3.8

Released 2023-08-20

- Drop Python 3.9 support. (#302)
- Use `concurrent.futures` instead of `multiprocessing.dummy` when *updating feeds* in parallel; `multiprocessing.dummy` does not work on some environments (e.g. AWS Lambda).
- Wrap unexpected hook errors in *UpdateHookError* instead of letting them bubble up, so plugin-raised exceptions for one feed don't prevent updates for the others during *update_feeds_iter()* / *update_feeds()*. (#218)

  **Warning:** **This is a minor compatibility break**; it is considered acceptable, since it fixes a bug / unexpected behavior.

  - Add new exceptions *UpdateHookError*, *SingleUpdateHookError*, and *UpdateHookErrorGroup*.
  - Try to run all *after_entry_update_hooks*, *after_feed_update_hooks*, and *after_feeds_update_hooks*, don't stop after one fails.
- Add *UpdateError* as parent of all update-related exceptions. (#218)
  - Narrow down the error type of *UpdateResult.value* from *ReaderError* to *UpdateError*.
  - Make *ParseError* inherit from *UpdateError*.
  - Document *update_feeds_iter()*, *update_feeds()*, and *update_feed()* can raise *UpdateError*s (other than *UpdateHookError* and *ParseError*).
- Make *ReaderWarning* inherit from *ReaderError*.
- Include a diagram of the *Exception hierarchy* in the *API reference*.
- Add werkzeug dependency, instead of vendoring selected `werkzeug.http` utilities.
- Rework lazy imports introduced in *version 3.3*. (#316)
- Make *reader._parser* a package, and move parsing-related modules into it. (#316)

### 6.3.6 Version 3.7

Released 2023-07-15

**Attention:** This is the last release to support Python 3.9; see #302 for details.

- Support PyPy 3.10. (#302)
- Remove the *twitter* experimental plugin (deprecated in *3.6*). (#310)
- Remove the *tumblr_gdpr* experimental plugin (not needed since August 2020). (#315)

### 6.3.7 Version 3.6

Released 2023-06-16

- Add documentation on *How to contribute to reader* and a detailed *Roadmap*. Thanks to Katharine Jarmul for finally getting me to do this. (#60)
- Document the low-level `delete_entries()` storage method. (#301, #96)
- Update vendored `reader._http_utils` to werkzeug 2.3.5.
- Deprecate the *twitter* experimental plugin, since the Twitter API does not have a (useful) free tier anymore. (#310)

  **Attention:** The *twitter* plugin will be removed in version 3.7.

### 6.3.8 Version 3.5

Released 2023-03-19

- Make *Entry.important* an *optional* boolean defaulting to `None`, so one can express "explicitly unimportant" (*don't care*) by setting it to `False`. This replaces the semantics for *don't care* introduced in *version 2.2*. (#254)

  **Warning:** **This is a minor compatibility break**, and should mostly affect code that checks identity (`if entry.important is True: ...`); code that uses *important* in a boolean context (`if entry.important: ...`) should not be affected.

  - *Entry.important* values will be migrated as follows:

    ```
    if read and not important and important_modified:
        important = False
    elif not important:
        important = None
    else:
        important = important
    ```

  - The `important` argument of *get_entries()*, *search_entries()*, etc. can also take string literals for more precise filtering, see *TristateFilterInput*.
  - The *mark_as_read* plugin does not set *read_modified* and *important_modified* anymore.
  - The web app uses the new *don't care* semantics.

- *set_entry_read()* and *set_entry_important()* do not coerce the flag value to `bool` anymore, and require it to be `True` or `False` (or `None`).

### 6.3.9 Version 3.4

Released 2023-01-22

- Drop Python 3.8 support. (#298)
- Document the parser *Internal API*. (#235, #255)
- Fix `preview_feed_list` plugin, broken by *3.3* parser refactoring. (#299)

### 6.3.10 Version 3.3

Released 2022-12-19

This release marks *reader*'s 5th anniversary and its 2000th commit.

**Attention:** This is the last release to support Python 3.8; see #298 for details.

- Support Python 3.11. (#289)
- Postpone update-related imports until needed. Shortens time from process start to usable Reader instance by 3x (imports are 72% faster). (#297)
- Refactor parser internals. (#297)

  **Note:** Plugins using the (unstable) session hooks should replace:

  ```
  reader._parser.session_hooks.request.append(...)
  reader._parser.session_hooks.response.append(...)
  ```

  with:

  ```
  reader._parser.session_factory.request_hooks.append(...)
  reader._parser.session_factory.response_hooks.append(...)
  ```

- *twitter* plugin: don't fail when deserializing tweets with missing `edit_history_tweet_ids` (fails in tweepy 4.11, warns in tweepy >4.12).

### 6.3.11 Version 3.2

Released 2022-09-14

- *UpdatedFeed* changes: added field *unmodified* and property *total*; fields *new* and *modified* became optional. (#96)
- Fix bug in *entry_dedupe* causing updates to fail if there were multiple *new* duplicates of the same issue. (#292)
- Fix bug in *readtime* and *mark_as_read* causing updates to fail if an entry was deleted by another plugin. (#292)
- Fix bug in *mark_as_read* causing updates to fail if an entry had no title.

- In the CLI, don't suppress the traceback of *ReaderError*, since it would also suppress it for bugs.
- In the CLI, stop using deprecated `click.get_terminal_size()`.

### 6.3.12 Version 3.1

Released 2022-08-29

- Drop *readtime* plugin dependency on readtime (which has a transitive dependency on lxml, which does not always have PyPy Windows wheels on PyPI). The `readtime` extra is deprecated, but remains available to avoid breaking dependent packages. (#286)
- Sort entries by added date most of the time, with the exception of those imported on the first update. Previously, entries would be sorted by added only if they were published less than 7 days ago, causing entries that appear in the feed months after their published to never appear at the top (so the user would never see them). (#279)

### 6.3.13 Version 3.0

Released 2022-07-30

**Attention:** This release contains backwards incompatible changes.

- Remove old database migrations.

  Remove *mark_as_read* config tag name migration.

  If you are upgrading from *reader* 2.10 or newer, no action is required.

  **Attention:** If you are upgrading to *reader* 3.0 from a version **older than 2.10**, you must open your database with *reader* 2.10 or newer once, to run the removed migrations:

```
pip install 'reader>=2.10,<3' && \
python - db.sqlite << EOF
import sys
from reader import make_reader
from reader.plugins.mark_as_read import _migrate_pre_2_7_metadata as migrate_
↪mark_as_read

reader = make_reader(sys.argv[1])

for feed in reader.get_feeds():
    migrate_mark_as_read(reader, feed)

print("OK")

EOF
```

- Remove code that issued deprecation warnings in versions 2.* (#268):
  - `Reader.get_feed_metadata()`
  - `Reader.get_feed_metadata_item()`
  - `Reader.set_feed_metadata_item()`

  - Reader.delete_feed_metadata_item()
  - Reader.get_feed_tags()
  - Reader.add_feed_tag()
  - Reader.remove_feed_tag()
  - MetadataError
  - MetadataNotFoundError
  - FeedMetadataNotFoundError
  - EntryMetadataNotFoundError
  - the object_id property of data objects and related exceptions
- Make some of the parameters of the following positional-only (#268):
  - *Reader.add_feed()*: feed
  - *Reader.delete_feed()*: feed
  - *Reader.change_feed_url()*: old, new
  - *Reader.get_feed()*: feed, default
  - *Reader.set_feed_user_title()*: feed, title
  - *Reader.enable_feed_updates()*: feed
  - *Reader.disable_feed_updates()*: feed
  - *Reader.update_feed()*: feed
  - *Reader.get_entry()*: entry, default
  - *Reader.set_entry_read()*: entry, read
  - *Reader.mark_entry_as_read()*: entry
  - *Reader.mark_entry_as_unread()*: entry
  - *Reader.set_entry_important()*: entry, important
  - *Reader.mark_entry_as_important()*: entry
  - *Reader.mark_entry_as_unimportant()*: entry
  - *Reader.add_entry()*: entry
  - *Reader.delete_entry()*: entry
  - *Reader.search_entries()*: query
  - *Reader.search_entry_counts()*: query
  - *Reader.get_tags()*: resource
  - *Reader.get_tag_keys()*: resource
  - *Reader.get_tag()*: resource, key, default
  - *Reader.set_tag()*: resource, key, value
  - *Reader.delete_tag()*: resource, key
  - *Reader.make_reader_reserved_name()*: key
  - *Reader.make_plugin_reserved_name()*: plugin_name, key

  - *FeedError* (and subclasses): `url`
  - *EntryError* (and subclasses): `feed_url`, `entry_id`
  - *TagError* (and subclasses): `resource_id`, `key`
- In *make_reader()*, wrap exceptions raised during plugin initialization in new exception *PluginInitError* instead of letting them bubble up. (#268)
- Swap the order of the first two arguments of *TagError* (and subclasses); `TagError(key, resource_id, ...)` becomes `TagError(resource_id, key, ...)`. (#268)

### 6.3.14 Version 2.17

Released 2022-07-23

- Deprecate the `object_id` property of data objects in favor of new property *resource_id*. *resource_id* is the same as `object_id`, except for feeds and feed-related exceptions it is of type `tuple[str]` instead of `str`. `object_id` **will be removed in version 3.0**. (#266, #268)
- Do not attempt too hard to run `PRAGMA optimize` if the database is busy. Prevents rare "database is locked" errors when multiple threads using the same reader terminate at the same time. (#206)

### 6.3.15 Version 2.16

Released 2022-07-17

- Allow using a *Reader* object from multiple threads directly (do not require it to be used as a context manager anymore). (#206)
- Allow *Reader* objects to be reused after closing. (#206, #284)
- Allow calling *close()* from any thread. (#206)
- Allow using a *Reader* object from multiple asyncio tasks. (#206)

### 6.3.16 Version 2.15

Released 2022-07-08

- Allow using *Reader* objects from threads other than the creating thread. (#206)
- Allow using *Reader* objects as context managers. (#206)

### 6.3.17 Version 2.14

Released 2022-06-30

- Mark *reader* as providing type information. Previously, code importing from *reader* would fail type checking with `error: Skipping analyzing "reader": module is installed, but missing library stubs or py.typed marker`. (#280)
- Drop Python 3.7 support. (#278)
- Support PyPy 3.9.

### 6.3.18 Version 2.13

Released 2022-06-28

- Add the *twitter* experimental plugin, which allows using a Twitter account as a feed. (#271)
- Skip with a warning entries that have no <guid> or <link> in an RSS feed; only raise `ParseError` if *all* entries have a missing id. (Note that both Atom and JSON Feed entries are required to have an id by their respective specifications.) Thanks to Mirek Długosz for the issue and pull request. (#281)
- Add `ReaderWarning`.

### 6.3.19 Version 2.12

Released 2022-03-31

- Add the `readtime` *built-in* plugin, which stores the entry read time as a tag during feed update. (#275)
- Allow running arbitrary actions *once* before/after updating feeds via `before_feeds_update_hooks` / `after_feeds_update_hooks`.
- Add `Entry.get_content()` and `Content.is_html`.
- In the web app, use the read time provided by the `readtime` plugin, instead of calculating it on each page load. Speeds up the rendering of the entries page by 20-30%, hopefully winning back the time lost when the read time feature was first added in *2.6*. (#275)
- In the web app, also show the read time for search results.

### 6.3.20 Version 2.11

Released 2022-03-17

- Fix issue causing `make_reader()` to fail with message `database requirement error: required SQLite compile options missing: ['ENABLE_JSON1']` when using SQLite 3.38 or newer. (#273)

### 6.3.21 Version 2.10

Released 2022-03-12

- Support entry and global tags. (#272, #228, #267)
- Remove `get_tags()` support for the `(None,)` (any feed) and `None` (any resource) wildcard resource values.

  **Warning:** **This is a minor compatibility break**, but is unlikely to affect existing users; the usefulness of the wildcards was limited, because it was impossible to tell to which resource a (key, value) pair belongs.

- Allow passing a *(feed URL,)* 1-tuple anywhere a feed URL can be passed to a `Reader` method.
- Remove the `global_metadata` experimental plugin (superseded by global tags).
- In the web application, support editing entry and global metadata. Fix broken delete metadata button. Fix broken error flashing.

### 6.3.22 Version 2.9

Released 2022-02-07

- Decrease *update_feeds()* memory usage by ~35% (using the maxrss before the call as baseline; overall process maxrss decreases by ~20%). The improvement is not in *reader* code, but in feedparser; *reader* will temporarily vendor feedparser until the fix makes it upstream and is released on PyPI. (#265)
- In the web application, allow sorting feeds by the number of entries: important, unread, per day during the last 1, 3, 12 months. (#249, #245).

### 6.3.23 Version 2.8

Released 2022-01-22

- Add generic tag methods *get_tags()*, *get_tag_keys()*, *get_tag()*, *set_tag()*, and *delete_tag()*, providing a unified interface for accessing tags as key-value pairs. (#266)

  Add the *TagError*, *TagNotFoundError*, and *ResourceNotFoundError* exceptions.

- Deprecate feed-specific tag and metadata methods (#266):
  - `get_feed_metadata()`, use *get_tags()* instead
  - `get_feed_metadata_item()`, use *get_tag()* instead
  - `set_feed_metadata_item()`, use *set_tag()* instead
  - `delete_feed_metadata_item()`, use *delete_tag()* instead
  - `get_feed_tags()`, use *get_tag_keys()* instead
  - `add_feed_tag()`, use *set_tag()* instead
  - `remove_feed_tag()`, use *delete_tag()* instead

  Deprecate `MetadataError`, `MetadataNotFoundError`, and `FeedMetadataNotFoundError`.

  All deprecated methods/exceptions **will be removed in version 3.0**.

- Add the `missing_ok` argument to *delete_feed()* and *delete_entry()*.
- Add the `exist_ok` argument to *add_feed()*.
- In the web application, show maxrss when debug is enabled. (#269)
- In the web application, decrease memory usage of the entries page when there are a lot of entries (e.g. for 2.5k entries, maxrss decreased from 115 MiB to 75 MiB), at the expense of making "entries for feed" slightly slower. (#269)

### 6.3.24 Version 2.7

Released 2022-01-04

- Tags and metadata now share the same namespace. See the *Resource tags* user guide section for details. (#266)
- The *mark_as_read* plugin now uses the `.reader.mark-as-read` metadata for configuration. Feeds using the old metadata, `.reader.mark_as_read`, will be migrated automatically on update until *reader* 3.0.
- Allow running arbitrary actions before updating feeds via *before_feed_update_hooks*.
- Expose *reader.plugins.DEFAULT_PLUGINS*.
- Add the `global_metadata` experimental plugin.

### 6.3.25 Version 2.6

Released 2021-11-15

- Retrieve feeds in parallel, but parse them serially; previously, feeds would be parsed in parallel. Decreases Linux memory usage by ~20% when using `workers`; the macOS decrease is less notable. (#261)
- Allow *update_feeds()* and *update_feeds_iter()* to filter feeds by `feed`, `tags`, `broken`, and `updates_enabled`. (#193, #219, #220)
- Allow *get_feeds()* and *get_feed_counts()* to filter feeds by `new`. (#217)
- Reuse the requests session when retrieving feeds; previously, each feed would get its own session.
- Add support for CLI plugins.
- Add the *cli_status* experimental plugin.
- In the web application, show entry read time.

### 6.3.26 Version 2.5

Released 2021-10-28

- In *add_feed()* and *change_feed_url()*, validate if the current Reader configuration can handle the new feed URL; if not, raise *InvalidFeedURLError* (a `ValueError` subclass). (#155)

  **Warning:** **This is a minor compatibility break**; previously, `ValueError` would never be raised for `str` arguments. To get the previous behavior (no validation), use `allow_invalid_url=True`.

- Allow users to add entries to an existing feed through the new *add_entry()* method. Allow deleting user-added entries through *delete_entry()*. (#239)
- Add the *added* and *added_by* Entry attributes. (#239)
- *Entry.updated* is now `None` if missing in the feed (*updated* became optional in *version 2.0*). Use *updated_not_none* for the pre-2.5 behavior. Do not swap *Entry.published* with *Entry.updated* for RSS feeds where *updated* is missing. (#183)
- Support PyPy 3.8.
- Fix bug causing *read_modified* and *important_modified* to be reset to `None` when an entry is updated.
- Fix bug where deleting an entry and then adding it again (with the same id) would fail if search was enabled and *update_search()* was not run before adding the new entry.

### 6.3.27 Version 2.4

Released 2021-10-19

- Enable search by default. (#252)
  - Add the `search_enabled` *make_reader()* argument. By default, search is enabled on the first *update_search()* call; the previous behavior was to do nothing.
  - Always install the full-text search dependencies (previously optional). The `search` extra remains available to avoid breaking dependent packages.
- Add the *subtitle* and *version* Feed attributes. (#223)

- Change the *mark_as_read* plugin to also explicitly mark matching entries as unimportant, similar to how the *don't care* web application button works. (#260)
- In the web application, show the feed subtitle. (#223)

### 6.3.28 Version 2.3

Released 2021-10-11

- Support Python 3.10. (#248)
- *entry_dedupe* now deletes old duplicates instead of marking them as read/unimportant. (#140)

> **Note:** Please comment in #140 / open an issue if you were relying on the old behavior.

- Fix *entry_dedupe* bug introduced in 2.2, causing the newest read entry to be marked as unread if none of its duplicates are read (idem for important). This was an issue *only when re-running the plugin for existing entries*, not for new entries (since new entries are unread/unimportant).

### 6.3.29 Version 2.2

Released 2021-10-08

- *entry_dedupe* plugin improvements: reduce false negatives by using approximate content matching, and make it possible to re-run the plugin for existing entries. (#202)
- Allow running arbitrary actions for updated feeds via *after_feed_update_hooks*. (#202)
- Add *set_entry_read()* and *set_entry_important()* to allow marking an entry as (un)read/(un)important through a boolean flag. (#256)
- Record when an entry is marked as read/important, and make it available through *read_modified* and *important_modified*. Allow providing a custom value using the `modified` argument of *set_entry_read()* and *set_entry_important()*. (#254)
- Make *entry_dedupe* copy *read_modified* and *important_modified* from the duplicates to the new entry. (#254)
- In the web application, allow marking an entry as *don't care* (read + unimportant explicitly set by the user) with a single button. (#254)
- In the web application, show the entry read modified / important modified timestamps as button tooltips. (#254)

### 6.3.30 Version 2.1

Released 2021-08-18

- Return *entry averages* for the past 1, 3, 12 months from the entry count methods. (#249)
- Use an index for `get_entry_counts(feed=...)` calls. Makes the /feeds?counts=yes page load 2-4x faster. (#251)
- Add *UpdateResult* *updated_feed*, *error*, and *not_modified* convenience properties. (#204)
- In the web application, show the feed entry count averages as a bar sparkline. (#249)

- Make the minimum SQLite version and required SQLite compile options `reader._storage` module globals, for easier monkeypatching. (#163)

  This is allows supplying a user-defined `json_array_length` function on platforms where SQLite doesn't come with the JSON1 extension (e.g. on Windows with stock Python earlier than 3.9; details).

  Note these globals are private, and thus *not* covered by the *backwards compatibility policy*.

### 6.3.31 Version 2.0

Released 2021-07-17

**Attention:** This release contains backwards incompatible changes.

- Remove old database migrations.

  If you are upgrading from *reader* 1.15 or newer, no action is required.

  **Attention:** If you are upgrading to *reader* 2.0 from a version **older than 1.15**, you must open your database with *reader* 1.15 or newer once, to run the removed migrations:

  ```
  pip install 'reader>=1.15,<2' && \
  python - db.sqlite << EOF
  import sys
  from reader import make_reader
  make_reader(sys.argv[1])
  print("OK")
  EOF
  ```

- Remove code that issued deprecation warnings in versions 1.* (#183):
  - `Reader.remove_feed()`
  - `Reader.mark_as_read()`
  - `Reader.mark_as_unread()`
  - `Reader.mark_as_important()`
  - `Reader.mark_as_unimportant()`
  - `Reader.iter_feed_metadata()`
  - the `get_feed_metadata(feed, key, default=no value, /)` form of `Reader.get_feed_metadata()`
  - `Reader.set_feed_metadata()`
  - `Reader.delete_feed_metadata()`
  - the `new_only` parameter of *`update_feeds()`* and *`update_feeds_iter()`*
  - `EntryError.url`
  - `UpdatedFeed.updated`
- The `datetime` attributes of *`Feed`* and *`Entry`* objects are now timezone-aware, with the timezone set to `utc`. Previously, they were naive datetimes representing UTC times. (#233)

- The parameters of *update_feeds()* and *update_feeds_iter()* are now keyword-only. (#183)
- The `feed_root` argument of *make_reader()* now defaults to `None` (don't open local feeds) instead of `''` (full filesystem access).
- *make_reader()* may now raise any *ReaderError*, not just *StorageError*.
- *Entry.updated* may now be `None`; use *updated_not_none* for the pre-2.0 behavior.

### 6.3.32 Version 1.20

Released 2021-07-12

- Add *after_entry_update_hooks*, which allow running arbitrary actions for updated entries. Thanks to Mirek Długosz for the issue and pull request. (#241)
- Raise *StorageError* when opening / operating on an invalid database, instead of a plain `sqlite3.DatabaseError`. (#243)

### 6.3.33 Version 1.19

Released 2021-06-16

- Drop Python 3.6 support. (#237)
- Support PyPy 3.7. (#234)
- Skip enclosures with no `href`/`url`; previously, they would result in a parse error. (#240)
- Stop using Travis CI (only use GitHub Actions). (#199)
- Add the `new` argument to *update_feeds()* and *update_feeds_iter()*; `new_only` is deprecated and **will be removed in 2.0**. (#217)
- Rename `UpdatedFeed.updated` to *modified*; for backwards compatibility, the old attribute will be available as a property **until version 2.0**, when it **will be removed.**. (#241)

> **Warning:** The signature of *UpdatedFeed* changed from `UpdatedFeed(url, new, updated)` to `UpdatedFeed(url, new, modified)`.
>
> **This is a minor compatibility break**, but only affects third-party code that instantiates UpdatedFeed *directly* with `updated` as a *keyword argument*.

### 6.3.34 Version 1.18

Released 2021-06-03

- Rename *Reader* feed metadata methods:
  - `iter_feed_metadata()` to `get_feed_metadata()`
  - `get_feed_metadata()` to `get_feed_metadata_item()`
  - `set_feed_metadata()` to `set_feed_metadata_item()`
  - `delete_feed_metadata()` to `delete_feed_metadata_item()`

For backwards compatibility, the old method signatures will continue to work **until version 2.0**, when they **will be removed.** (#183)

> **Warning:** The `get_feed_metadata(feed, key[, default]) -> value` form is backwards-compatible *only when the arguments are positional*.
>
> **This is a minor compatibility break**; the following work in 1.17, but do not in 1.18:
>
> ```
> # raises TypeError
> reader.get_feed_metadata(feed, key, default=None)
>
> # returns `(key, value), ...` instead of `value`
> reader.get_feed_metadata(feed, key=key)
> ```
>
> The pre-1.18 `get_feed_metadata()` (1.18 `get_feed_metadata_item()`) is intended to have positional-only arguments, but this cannot be expressed easily until Python 3.8.

- Rename `MetadataNotFoundError` to `FeedMetadataNotFoundError`. `MetadataNotFoundError` remains available, and is a superclass of `FeedMetadataNotFoundError` for backwards compatibility. (#228)

> **Warning:** The signatures of the following exceptions changed:
>
> `MetadataError`
> : Takes a new required `key` argument, instead of no required arguments.
>
> `MetadataNotFoundError`
> : Takes only one required argument, `key`; the `url` argument has been removed.
>
> Use `FeedMetadataNotFoundError` instead.
>
> **This is a minor compatibility break**, but only affects third-party code that instantiates these exceptions *directly*.

- Rename `EntryError.url` to `feed_url`; for backwards compatibility, the old attribute will be available as a property **until version 2.0**, when it **will be removed.** (#183).

> **Warning:** The signature of *EntryError* (and its subclasses) changed from `EntryError(url, id)` to `EntryError(feed_url, id)`.
>
> **This is a minor compatibility break**, but only affects third-party code that instantiates these exceptions *directly* with `url` as a *keyword argument*.

- Rename `remove_feed()` to *delete_feed()*. For backwards compatibility, the old method will continue to work **until version 2.0**, when it **will be removed.** (#183)
- Rename *Reader* `mark_as_...` methods:
  - `mark_as_read()` to *mark_entry_as_read()*
  - `mark_as_unread()` to *mark_entry_as_unread()*
  - `mark_as_important()` to *mark_entry_as_important()*
  - `mark_as_unimportant()` to *mark_entry_as_unimportant()*

  For backwards compatibility, the old methods will continue to work **until version 2.0**, when they **will be removed.** (#183)

- Fix feeds with no title sometimes missing from the *get_feeds()* results when there are more than 256 feeds (`Storage.chunk_size`). (#203)
- When serving the web application with `python -m reader serve`, don't set the `Referer` header for cross-origin requests. (#209)

### 6.3.35 Version 1.17

Released 2021-05-06

- Reserve tags and metadata keys starting with `.reader.` and `.plugin.` for *reader*- and plugin-specific uses. See the *Reserved names* user guide section for details. (#186)
- Ignore *updated* when updating feeds; only update the feed if other feed data changed or if any entries were added/updated. (#231)

  Prevents spurious updates for feeds whose *updated* changes excessively (either because the entries' content changes excessively, or because an RSS feed does not have a `dc:date` element, and feedparser falls back to `lastBuildDate` for *updated*).
- The `regex_mark_as_read` experimental plugin is now *built-in*. To use it with the CLI / web application, use the plugin name instead of the entry point (`reader.mark_as_read`).

  The config metadata key and format changed; the config will be migrated automatically on the next feed update, **during reader version 1.17 only**. If you used `regex_mark_as_read` and are upgrading to a version >1.17, install 1.17 (`pip install reader==1.17`) and run a full feed update (`python -m reader update`) before installing the newer version.
- The `enclosure-tags`, `preview-feed-list`, and `sqlite-releases` unstable extras are not available anymore. Use the `unstable-plugins` extra to install dependencies of the unstable plugins instead.
- In the web application, allow updating a feed manually. (#195)

### 6.3.36 Version 1.16

Released 2021-03-29

- Allow *make_reader()* to load plugins through the `plugins` argument. (#229)

  Enable the *ua_fallback* plugin by default.

  *make_reader()* may now raise *InvalidPluginError* (a `ValueError` subclass, which it already raises implicitly) for invalid plugin names.
- The `enclosure_dedupe`, `feed_entry_dedupe`, and `ua_fallback` plugins are now *built-in*. (#229)

  To use them with the CLI / web application, use the plugin name instead of the entry point:

```
reader._plugins.enclosure_dedupe:enclosure_dedupe   -> reader.enclosure_dedupe
reader._plugins.feed_entry_dedupe:feed_entry_dedupe -> reader.entry_dedupe
reader._plugins.ua_fallback:init                    -> reader.ua_fallback
```

- Remove the `plugins` extra; plugin loading machinery does not have additional dependencies anymore.
- Mention in the *User guide* that all *reader* functions/methods can raise `ValueError` or `TypeError` if passed invalid arguments. There is no behavior change, this is just documenting existing, previously undocumented behavior.

### 6.3.37 Version 1.15

Released 2021-03-21

- Update entries whenever their content changes, regardless of their *updated* date. (#179)

  Limit content-only updates (not due to an *updated* change) to 24 consecutive updates, to prevent spurious updates for entries whose content changes excessively (for example, because it includes the current time). (#225)

  Previously, entries would be updated only if the entry *updated* was *newer* than the stored one.

- Fix bug causing entries that don't have *updated* set in the feed to not be updated if the feed is marked as stale. Feed staleness is an internal feature used during storage migrations; this bug could only manifest when migrating from 0.22 to 1.x. (found during #179)
- Minor web application improvements.
- Minor CLI improvements.

### 6.3.38 Version 1.14

Released 2021-02-22

- Add the *update_feeds_iter()* method, which yields the update status of each feed as it gets updated. (#204)
- Change the return type of *update_feed()* from `None` to `Optional[UpdatedFeed]`. (#204)
- Add the `session_timeout` argument to *make_reader()* to set a timeout for retrieving HTTP(S) feeds. The default (connect timeout, read timeout) is (3.05, 60) seconds; the previous behavior was to *never time out*.
- Use `PRAGMA user_version` instead of a version table. (#210)
- Use `PRAGMA application_id` to identify reader databases; the id is `0x66656564` – `read` in ASCII / UTF-8. (#211)
- Change the `reader update` command to show a progress bar and update summary (with colors), instead of plain log output. (#204)
- Fix broken Mypy config following 0.800 release. (#213)

### 6.3.39 Version 1.13

Released 2021-01-29

- JSON Feed support. (#206)
- Split feed retrieval from parsing; should make it easier to add new/custom parsers. (#206)
- Prevent any logging output from the `reader` logger by default. (#207)
- In the `preview_feed_list` plugin, add `<link rel=alternative ...>` tags as a feed detection heuristic.
- In the `preview_feed_list` plugin, add `<a>` tags as a *fallback* feed detection heuristic.
- In the web application, fix bug causing the entries page to crash when counts are enabled.

### 6.3.40 Version 1.12

Released 2020-12-13

- Add the `limit` and `starting_after` arguments to *get_feeds()*, *get_entries()*, and *search_entries()*, allowing them to be used in a paginated fashion. (#196)
- Add the `object_id` property that allows getting the unique identifier of a data object in a uniform way. (#196)
- In the web application, add links to toggle feed/entry counts. (#185)

### 6.3.41 Version 1.11

Released 2020-11-28

- Allow disabling feed updates for specific feeds. (#187)
- Add methods to get aggregated feed and entry counts. (#185)
- In the web application: allow disabling feed updates for a feed; allow filtering feeds by whether they have updates enabled; do not show feed update errors for feeds that have updates disabled. (#187)
- In the web application, show feed and entry counts when `?counts=yes` is used. (#185)
- In the web application, use YAML instead of JSON for the tags and metadata fields.

### 6.3.42 Version 1.10

Released 2020-11-20

- Use indexes for *get_entries()* (recent order); should make calls 10-30% faster. (#134)
- Allow sorting *search_entries()* results randomly. Allow sorting search results randomly in the web application. (#200)
- Reraise unexpected errors caused by parser bugs instead of replacing them with an `AssertionError`.
- Add the `sqlite_releases` custom parser plugin.
- Refactor the HTTP feed sub-parser to allow reuse by custom parsers.
- Add a user guide, and improve other parts of the documentation. (#194)

### 6.3.43 Version 1.9

Released 2020-10-28

- Support Python 3.9. (#199)
- Support Windows (requires Python >= 3.9). (#163)
- Use GitHub Actions to do macOS and Windows CI builds. (#199)
- Rename the `cloudflare_ua_fix` plugin to `ua_fallback`. Retry any feed that gets a 403, not just those served by Cloudflare. (#181)
- Fix type annotation to avoid mypy 0.790 errors. (#198)

### 6.3.44 Version 1.8

Released 2020-10-02

- Drop feedparser 5.x support (deprecated in 1.7); use feedparser 6.x instead. (#190)
- Make the string representation of *ReaderError* and its subclasses more consistent; add error messages and improve the existing ones. (#173)
- Add method *change_feed_url()* to change the URL of a feed. (#149)
- Allow changing the URL of a feed in the web application. (#149)
- Add more tag navigation links to the web application. (#184)
- In the `feed_entry_dedupe` plugin, copy the important flag from the old entry to the new one. (#140)

### 6.3.45 Version 1.7

Released 2020-09-19

- Add new methods to support feed tags: `add_feed_tag()`, `remove_feed_tag()`, and `get_feed_tags()`. Allow filtering feeds and entries by their feed tags. (#184)
- Add the `broken` argument to *get_feeds()*, which allows getting only feeds that failed / did not fail during the last update. (#189)
- feedparser 5.x support is deprecated in favor of feedparser 6.x. Using feedparser 5.x will raise a deprecation warning in version 1.7, and support will be removed the following version. (#190)
- Tag-related web application features: show tags in the feed list; allow adding/removing tags; allow filtering feeds and entries by their feed tag; add a page that lists all tags. (#184)
- In the web application, allow showing only feeds that failed / did not fail. (#189)
- In the `preview_feed_list` plugin, add `<meta>` tags as a feed detection heuristic.
- Add a few property-based tests. (#188)

### 6.3.46 Version 1.6

Released 2020-09-04

- Add the `feed_root` argument to *make_reader()*, which allows limiting local feed parsing to a specific directory or disabling it altogether. Using it is recommended, since by default *reader* will access any local feed path (in 2.0, local file parsing will be disabled by default). (#155)
- Support loading CLI and web application settings from a *configuration file*. (#177)
- Fail fast for feeds that return HTTP 4xx or 5xx status codes, instead of (likely) failing later with an ambiguous XML parsing error. The cause of the raised *ParseError* is now an instance of `requests.HTTPError`. (#182)
- Add `cloudflare_ua_fix` plugin (work around Cloudflare sometimes blocking requests). (#181)
- feedparser 6.0 (beta) compatibility fixes.
- Internal parser API changes to support alternative parsers, pre-request hooks, and making arbitrary HTTP requests using the same logic *Reader* uses. (#155)
- In the /preview page and the `preview_feed_list` plugin, use the same plugins the main *Reader* does. (enabled by #155)

### 6.3.47 Version 1.5

Released 2020-07-30

- Use rowid when deleting from the search index, instead of the entry id. Previously, each *update_search()* call would result in a full scan, even if there was nothing to update/delete. This should reduce the amount of reads significantly (deleting 4 entries from a database with 10k entries resulted in an 1000x decrease in bytes read). (#178)
- Require at least SQLite 3.18 (released 2017-03-30) for the current *update_search()* implementation; all other *reader* features continue to work with SQLite >= 3.15. (#178)
- Run `PRAGMA optimize` on *close()*. This should increase the performance of all methods. As an example, in #178 it was found that *update_search()* resulted in a full scan of the entries table, even if there was nothing to update; this change should prevent this from happening. (#143)

---

**Note:** `PRAGMA optimize` is a no-op in SQLite versions earlier than 3.18. In order to avoid the case described above, you should run ANALYZE regularly (e.g. every few days).

---

### 6.3.48 Version 1.4

Released 2020-07-13

- Work to reduce the likelihood of "database is locked" errors during updates (#175):
  - Prepare entries to be added to the search index (*update_search()*) outside transactions.
  - Fix bug causing duplicate rows in the search index when an entry changes while updating the search index.
  - Update the search index only when the indexed values change (details below).
  - Use SQLite WAL (details below).
- Update the search index only when the indexed values change. Previously, any change on a feed would result in all its entries being re-indexed, even if the feed title or the entry content didn't change. This should reduce the *update_search()* run time significantly.
- Use SQLite's write-ahead logging to increase concurrency. At the moment there is no way to disable WAL. This change may be reverted in the future. (#169)
- Require at least click 7.0 for the `cli` extra.
- Do not fail for feeds with incorrectly-declared media types, if feedparser can parse the feed; this is similar to the current behavior for incorrectly-declared encodings. (#171)
- Raise *ParseError* during update for feeds feedparser can't detect the type of, instead of silently returning an empty feed. (#171)
- Add `sort` argument to *search_entries()*. Allow sorting search results by recency in addition to relevance (the default). (#176)
- In the web application, display a nice error message for invalid search queries instead of returning an HTTP 500 Internal Server Error.
- Other minor web application improvements.
- Minor CLI logging improvements.

### 6.3.49 Version 1.3

Released 2020-06-23

- If a feed failed to update, provide details about the error in *Feed.last_exception*. (#68)
- Show details about feed update errors in the web application. (#68)
- Expose the *added* and *last_updated* Feed attributes.
- Expose the *last_updated* Entry attribute.
- Raise *ParseError* / log during update if an entry has no id, instead of unconditionally raising AttributeError. (#170)
- Fall back to <link> as entry id if an entry in an RSS feed has no <guid>; previously, feeds like this would fail on update. (#170)
- Minor web application improvements (show feed added/updated date).
- In the web application, handle previewing an invalid feed nicely instead of returning an HTTP 500 Internal Server Error. (#172)
- Internal API changes to support multiple storage implementations in the future. (#168)

### 6.3.50 Version 1.2

Released 2020-05-18

- Minor web application improvements.
- Remove unneeded additional query in methods that use pagination (for n = len(result) / page size, always do n queries instead n+1). *get_entries()* and *search_entries()* are now 33–7% and 46–36% faster, respectively, for results of size 32–256. (#166)
- All queries are now chunked/paginated to avoid locking the SQLite storage for too long, decreasing the chance of concurrent queries timing out; the problem was most visible during *update_search()*. This should cap memory usage for methods returning an iterable that were not paginated before; previously the whole result set would be read before returning it. (#167)

### 6.3.51 Version 1.1

Released 2020-05-08

- Add `sort` argument to *get_entries()*. Allow sorting entries randomly in addition to the default most-recent-first order. (#105)
- Allow changing the entry sort order in the web application. (#105)
- Use a query builder instead of appending strings manually for the more complicated queries in search and storage. (#123)
- Make searching entries faster by filtering them *before* searching; e.g. if 1/5 of the entries are read, searching only read entries is now ~5x faster. (enabled by #123)

### 6.3.52 Version 1.0.1

Released 2020-04-30

- Fix bug introduced in *0.20* causing *update_feeds()* to silently stop updating the remaining feeds after a feed failed. (#164)

### 6.3.53 Version 1.0

Released 2020-04-28

- Make all private submodules explicitly private. (#156)

  **Note:** All direct imports from *reader* continue to work.

  - The `reader.core.*` modules moved to `reader.*` (most of them prefixed by _).
  - The web application WSGI entry point moved from `reader.app.wsgi:app` to `reader._app.wsgi:app`.
  - The entry points for plugins that ship with reader moved from `reader.plugins.*` to `reader._plugins.*`.

- Require at least beautifulsoup4 4.5 for the `search` extra (before, the version was unspecified). (#161)
- Rename the web application dependencies extra from `web-app` to `app`.
- Fix relative link resolution and content sanitization; sgmllib3k is now a required dependency for this reason. (#125, #157)

### 6.3.54 Version 0.22

Released 2020-04-14

- Add the *Entry.feed_url* attribute. (#159)
- Rename the *EntrySearchResult* `feed` attribute to *feed_url*. Using `feed` will raise a deprecation warning in version 0.22, and will be removed in the following version. (#159)
- Use `executemany()` instead of `execute()` in the SQLite storage. Makes updating feeds (excluding network calls) 5-10% faster. (#144)
- In the web app, redirect to the feed's page after adding a feed. (#119)
- In the web app, show highlighted search result snippets. (#122)

### 6.3.55 Version 0.21

Released 2020-04-04

- Minor consistency improvements to the web app search button. (#122)
- Add support for web application plugins. (#80)
- The enclosure tag proxy is now a plugin, and is disabled by default. See its documentation for details. (#52)
- In the web app, the "add feed" button shows a preview before adding the feed. (#145)
- In the web app, if the feed to be previewed is not actually a feed, show a list of feeds linked from that URL. This is a plugin, and is disabled by default. (#150)

- reader now uses a User-Agent header like `python-reader/0.21` when retrieving feeds instead of the default requests one. (#154)

### 6.3.56 Version 0.20

Released 2020-03-31

- Fix bug in *enable_search()* that caused it to fail if search was already enabled and the reader had any entries.
- Add an `entry` argument to *get_entries()*, for symmetry with *search_entries()*.
- Add a `feed` argument to *get_feeds()*.
- Add a `key` argument to `get_feed_metadata()`.
- Require at least requests 2.18 (before, the version was unspecified).
- Allow updating feeds concurrently; add a `workers` argument to *update_feeds()*. (#152)

### 6.3.57 Version 0.19

Released 2020-03-25

- Support PyPy 3.6.
- Allow *searching for entries*. (#122)
- Stricter type checking for the core modules.
- Various changes to the storage internal API.

### 6.3.58 Version 0.18

Released 2020-01-26

- Support Python 3.8.
- Increase the *get_entries()* recent threshold from 3 to 7 days. (#141)
- Enforce type checking for the core modules. (#132)
- Use dataclasses for the data objects instead of attrs. (#137)

### 6.3.59 Version 0.17

Released 2019-10-12

- Remove the `which` argument of *get_entries()*. (#136)
- *Reader* objects should now be created using *make_reader()*. Instantiating Reader directly will raise a deprecation warning.
- The resources associated with a reader can now be released explicitly by calling its *close()* method. (#139)
- Make the database schema more strict regarding nulls. (#138)
- Tests are now run in a random order. (#142)

### 6.3.60 Version 0.16

Released 2019-09-02

- Allow marking entries as important. (#127)
- *get_entries()* and *get_feeds()* now take only keyword arguments.
- *get_entries()* argument `which` is now deprecated in favor of `read`. (#136)

### 6.3.61 Version 0.15

Released 2019-08-24

- Improve entry page rendering for text/plain content. (#117)
- Improve entry page rendering for images and code blocks. (#126)
- Show enclosures on the entry page. (#128)
- Show the entry author. (#129)
- Fix bug causing the enclosure tag proxy to use too much memory. (#133)
- Start using mypy on the core modules. (#132)

### 6.3.62 Version 0.14

Released 2019-08-12

- Drop Python 3.5 support. (#124)
- Improve entry ordering implementation. (#110)

### 6.3.63 Version 0.13

Released 2019-07-12

- Add entry page. (#117)
- *get_feed()* now raises *FeedNotFoundError* if the feed does not exist; use `get_feed(..., default=None)` for the old behavior.
- Add *get_entry()*. (#120)

### 6.3.64 Version 0.12

Released 2019-06-22

- Fix flashed messages never disappearing. (#81)
- Minor metadata page UI improvements.
- Allow limiting the number of entries on the entries page via the `limit` URL parameter.
- Add link to the feed on the entries and feeds pages. (#118)
- Use Black and pre-commit to enforce style.

### 6.3.65 Version 0.11

Released 2019-05-26

- Support storing per-feed metadata. (#114)
- Add feed metadata page to the web app. (#114)
- The `regex_mark_as_read` plugin is now configurable via feed metadata; drop support for the `READER_PLUGIN_REGEX_MARK_AS_READ_CONFIG` file. (#114)

### 6.3.66 Version 0.10

Released 2019-05-18

- Unify plugin loading and error handling code. (#112)
- Minor improvements to CLI error reporting.

### 6.3.67 Version 0.9

Released 2019-05-12

- Improve the *get_entries()* sorting algorithm. Fixes a bug introduced by #106 (entries of new feeds would always show up at the top). (#113)

### 6.3.68 Version 0.8

Released 2019-04-21

- Make the internal APIs use explicit types instead of tuples. (#111)
- Finish updater internal API. (#107)
- Automate part of the release process (`scripts/release.py`).

### 6.3.69 Version 0.7

Released 2019-04-14

- Increase timeout of the button actions from 2 to 10 seconds.
- *get_entries()* now sorts entries by the import date first, and then by *published*/*updated*. (#106)
- Add `enclosure_dedupe` plugin (deduplicate enclosures of an entry). (#78)
- The `serve` command now supports loading plugins. (#78)
- `reader.app.wsgi` now supports loading plugins. (#78)

### 6.3.70 Version 0.6

Released 2019-04-13

- Minor web application style changes to make the layout more condensed.
- Factor out update logic into a separate interface. (#107)
- Fix update failing if the feed does not have a content type header. (#108)

### 6.3.71 Version 0.5

Released 2019-02-09

- Make updating new feeds up to 2 orders of magnitude faster; fixes a problem introduced by #94. (#104)
- Move the core modules to a separate subpackage and enforce test coverage (`make coverage` now fails if the coverage for core modules is less than 100%). (#101)
- Support Python 3.8 development branch.
- Add `dev` and `docs` extras (to install development requirements).
- Build HTML documentation when running tox.
- Add `test-all` and `docs` make targets (to run tox / build HTML docs).

### 6.3.72 Version 0.4

Released 2019-01-02

- Support Python 3.7.
- Entry *content* and *enclosures* now default to an empty tuple instead of `None`. (#99)
- *get_feeds()* now sorts feeds by *user_title* or *title* instead of just *title*. (#102)
- *get_feeds()* now sorts feeds in a case insensitive way. (#103)
- Add `sort` argument to *get_feeds()*; allows sorting feeds by title or by when they were added. (#98)
- Allow changing the feed sort order in the web application. (#98)

### 6.3.73 Version 0.3

Released on 2018-12-22

- *get_entries()* now prefers sorting by *published* (if present) to sorting by *updated*. (#97)
- Add `regex_mark_as_read` plugin (mark new entries as read based on a regex). (#79)
- Add `feed_entry_dedupe` plugin (deduplicate new entries for a feed). (#79)
- Plugin loading machinery dependencies are now installed via the `plugins` extra.
- Add a plugins section to the documentation.

### 6.3.74 Version 0.2

Released on 2018-11-25

- Factor out storage-related functionality into a separate interface. (#94)
- Fix `update --new-only` updating the same feed repeatedly on databases that predate `--new-only`. (#95)
- Add web application screenshots to the documentation.

### 6.3.75 Version 0.1.1

Released on 2018-10-21

- Fix broken `reader serve` command (broken in 0.1).
- Raise *StorageError* for unsupported SQLite configurations at *Reader* instantiation instead of failing at runtime with a generic `StorageError("sqlite3 error")`. (#92)
- Fix wrong submit button being used when pressing enter in non-button fields. (#69)
- Raise *StorageError* for failed migrations instead of an undocumented exception. (#92)
- Use `requests-mock` in parser tests instead of a web server (test suite run time down by ~35%). (#90)

### 6.3.76 Version 0.1

Released on 2018-09-15

- Initial release; public API stable.
- Support broken Tumblr feeds via the the *tumblr_gdpr* plugin. (#67)

CHAPTER

SEVEN

# INDICES AND TABLES

- genindex
- search

# PYTHON MODULE INDEX

## r

# INDEX

## Symbols



## A

## G



## H



## I



## L

## M



## N



## O



## P



## Q



## R

## S

## T



## U



## V

## W