reader Documentation

Release 1.20

lemon24

Jul 17, 2021

CONTENTS

1 Features **3**

2 Quickstart **5**

3 User guide **7**
- 3.1 Why *reader*? . . . 7
- 3.2 Installation . . . 8
- 3.3 Tutorial . . . 9
- 3.4 User guide . . . 14

4 API reference **25**
- 4.1 API reference . . . 25

5 Unstable features **49**
- 5.1 Command-line interface . . . 49
- 5.2 Web application . . . 55
- 5.3 Configuration . . . 56
- 5.4 Plugins . . . 62

6 Project information **67**
- 6.1 Backwards compatibility . . . 67
- 6.2 Development . . . 68
- 6.3 Changelog . . . 76

7 Indices and tables **91**

Python Module Index **93**

Index **95**

reader is a Python feed reader library.

It aims to allow writing feed reader applications without any business code, and without enforcing a dependency on a particular framework.

CHAPTER

ONE

FEATURES

reader allows you to:

- retrieve, store, and manage **Atom**, **RSS**, and **JSON** feeds
- mark entries as read or important
- add tags and metadata to feeds
- filter feeds and articles
- full-text search articles
- 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[search]
```

```
>>> 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_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.enable_search()
>>> reader.update_search()
>>>
>>> for e in list(reader.search_entries('year'))[: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?

… while still supporting all the feed types feedparser does?

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

3.1.2 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.

3.1.3 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 3.7 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.
- iso8601 parses dates in ISO 8601 / RFC 3339; used for JSON Feed parsing.

reader also depends on the `sqlite3` standard library module (at least SQLite 3.15), and on the JSON1 SQLite extension.

Note: reader works out of the box on Windows only starting with Python 3.9, because the SQLite bundled with the official Python distribution does **not** include the JSON1 extension in earlier versions. That said, it should be possible to build `sqlite3` with a newer version of SQLite; see #163 for details.

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 1.20, *reader* has the following extras:

- `search` provides *full-text search* functionality; search also requires that the SQLite used by `sqlite3` was compiled with the FTS5 extension, and is at least version 3.18.
- `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, FeedExistsError

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

reader = make_reader("db.sqlite")

def add_and_update_feed():
    try:
        reader.add_feed(feed_url)
    except FeedExistsError:
        pass
    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)
```

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
# 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:
```

(continues on next page)

(continued from previous page)

```
        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)

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

The *Reader* object persists feed and entry state and provides operations on them.

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.

After you are done with the reader, call *close()* to release the resources associated with it:

```
>>> reader.close()
```

While the same thing will eventually happen when the reader is garbage-collected, it is recommended to call *close()* explicitly, especially in long-running processes or when you create multiple readers pointing to the same database. You can use `contextlib.closing()` to do this automatically:

```
>>> from contextlib import closing
>>> with closing(make_reader('db.sqlite')) as reader:
...     ... # do stuff with reader
...
```

3.4.2 File-system access

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

For security reasons, you might want to restrict file-system access to a single directory or prevent it entirely; you can do so by using the `feed_root` *make_reader()* argument:

```
>>> # 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 on update, resolves to /feed.xml, which is above /feeds
>>> reader.add_feed("file:../feed.xml")
>>> # all local paths will fail to update
>>> reader = make_reader("db.sqlite", feed_root=None)
```

Note that it is still possible to *add* local feeds regardless of `feed_root`; it is *updating* them that will fail.

3.4.3 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, ...)
```

3.4.4 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), 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
    ),
    title='Hello Internet',
    link='http://www.hellointernet.fm/',
    author='CGP Grey',
    added=datetime.datetime(2020, 10, 12),
    last_updated=datetime.datetime(2020, 10, 12)
)
```

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
Hello Internet by CGP Grey, updated on 2020-02-28 09:34:02
```

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 duplicate entries by using a *plugin* like `feed_entry_dedupe`.

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 as *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.feed.title, '-', entry.title)
...
Hello Internet - H.I. #135: Place Your Bets
Hello Internet - # H.I. 134: Boxing Day
```

3.4.11 Full-text search

Note: The search functionality is optional, use the `search` extra to install its *dependencies*.

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

Since search adds some overhead, it needs to be enabled by calling *enable_search()* (this is persistent across Reader instances using the same database, and only needs to be done once). Also, the search index must be kept in sync by calling *update_search()* regularly (usually after updating the feeds).

```
>>> reader.enable_search()
>>> 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, which contain snippets of relevant entry/feed fields, with the parts that matched highlighted.

By default, the results are filtered by relevance; you can sort them most-recent first by passing `sort='recent'`.

search_entries() allows filtering the results just as *get_entries()* does.

3.4.12 Feed metadata

Feeds can have metadata, key-value pairs where the values are any JSON-serializable data:

```
>>> reader.get_feed_metadata_item(feed, 'key', 'default')
'default'
>>> reader.set_feed_metadata_item(feed, 'key', 'value')
>>> reader.get_feed_metadata_item(feed, 'key', 'default')
'value'
>>> reader.set_feed_metadata_item(feed, 'another', {'one': [2]})
>>> dict(reader.get_feed_metadata(feed))
{'another': {'one': [2]}, 'key': 'value'}
```

Common uses for metadata are plugin and UI settings.

Note that metadata keys and the top-level keys of dict metadata 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 character set.

3.4.13 Feed tags

Feeds can also have tags:

```
>>> reader.add_feed_tag(feed, 'one')
>>> reader.add_feed_tag(feed, 'two')
>>> set(reader.get_feed_tags(feed))
{'one', 'two'}
```

Tags can be used for filtering feeds and entries (see the *get_feeds()* documentation 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 tags 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 character set.

3.4.14 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=134, broken=3, updates_enabled=132)
>>> reader.get_entry_counts()
EntryCounts(total=11843, read=9762, important=45, has_enclosures=4273)
>>> reader.search_entry_counts('hello internet')
EntrySearchCounts(total=207, read=196, important=0, has_enclosures=172)
```

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 chain(reader.get_feed_tags(), [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: 29 feeds, 4277 entries
python: 29 feeds, 1281 entries
self: 2 feeds, 67 entries
tech: 79 feeds, 5527 entries
webcomic: 6 feeds, 1609 entries
<no tag>: 22 feeds, 1118 entries
```

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*. By default, only *reader.ua_fallback* is enabled.

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=[
...     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 `object_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:

- tags
- metadata keys
- the top-level keys of dict metadata 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` metadata.)

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, metadata 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 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=''*, *plugins=...*, *session_timeout=(3.05, 60)*, *reserved_name_scheme=...*)

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), '' (full filesystem access; default), 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. The only plugin used by default is reader.ua_fallback.
- **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) or None*) – Value for *reserved_name_scheme*. The prefixes default to .reader./.plugin., and the separator to .

Returns The reader.

Return type *Reader*

Raises

- ***StorageError*** –
- ***InvalidPluginError*** – If an invalid plugin name is passed to plugins.

New in version 1.6: The feed_root keyword argument.

Changed in version 2.0: The default feed_root behavior will change from *full filesystem access* ('') to *don't open local feeds* (None).

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

New in version 1.17: The reserved_name_scheme 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

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

Important: The *Reader* object is not thread safe; its methods should be called only from the thread that created it.

To access the same database from multiple threads, create one instance in each thread. If you have a strong use case preventing you to do so, please +1 / comment in #206.

New in version 1.13: JSON Feed support.

after_entry_update_hooks
List of functions called for each updated entry after the feed was 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 *object_id*; all other attributes may be missing (accessing them may raise `AttributeError`).

New in version 1.20.

close()
Close this *Reader*.

Releases any underlying resources associated with the reader.

The reader becomes unusable from this point forward; a *ReaderError* will be raised if any other method is called.

Raises ***ReaderError*** –

add_feed(*feed*)
Add a new feed.

Feed updates are enabled by default.

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

Raises

- ***FeedExistsError*** –
- ***StorageError*** –

delete_feed(*feed*)
Delete a feed and all of its entries, metadata, and tags.

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

Raises

- ***FeedNotFoundError*** –

- *StorageError* –

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

remove_feed(*feed*)
Deprecated alias for *delete_feed()*.

Deprecated since version 1.18: This method will be removed in *reader* 2.0. Use *delete_feed()* instead.

change_feed_url(*old*, *new*)
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* (except *updated*, which gets set to None). All other feed attributes are set to their default values.

The entries, tags and metadata are preserved.

Parameters

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

Raises

- *FeedNotFoundError* – If `old` does not exist.
- *FeedExistsError* – If `new` already exists.
- *StorageError* –

New in version 1.8.

get_feeds(*, *feed=None*, *tags=None*, *broken=None*, *updates_enabled=None*, *sort='title'*, *limit=None*, *starting_after=None*)
Get all or some of the feeds.

The `tags` argument can be a list of one or more feed 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 feeds 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

Parameters

- **feed** (*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.
- **broken** (*bool or None*) – Only return broken / healthy feeds.
- **updates_enabled** (*bool or None*) – Only return feeds that have updates enabled / disabled.
- **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 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 1.7: The `tags` keyword argument.

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

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

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

get_feed(*feed*, *default=no value*)

Get a feed.

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

Parameters

- **feed** (*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* –

get_feed_counts(*, *feed=None*, *tags=None*, *broken=None*, *updates_enabled=None*)
Count all or some of the feeds.

See *get_feeds()* for details on how filtering works.

Parameters

- **feed** (*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.
- **broken** (*bool or None*) – Only count broken / healthy feeds.
- **updates_enabled** (*bool or None*) – Only count feeds that have updates enabled / disabled.

Returns

Return type *FeedCounts*

Raises *StorageError* –

New in version 1.11.

set_feed_user_title(*feed*, *title*)
Set a user-defined title for a feed.

Parameters

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

Raises

- *FeedNotFoundError* –
- *StorageError* –

enable_feed_updates(*feed*)
Enable updates for a feed.

See *update_feeds()* for details.

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

Raises

- *FeedNotFoundError* –
- *StorageError* –

New in version 1.11.

disable_feed_updates(*feed*)
Disable updates for a feed.

See *update_feeds()* for details.

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

Raises

- *FeedNotFoundError* –
- *StorageError* –

New in version 1.11.

update_feeds(*new_only=no value*, *workers=1*, ***, *new=no value*)
Update all the feeds that have updates enabled.

Silently skip feeds that raise *ParseError*.

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

Parameters

- **new_only** (*bool*) – Only update feeds that have never been updated. Defaults to False.

  Deprecated since version 1.19: Use `new` instead.

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

Raises *StorageError* –

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

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.

Deprecated since version 1.19: The `new_only` argument (will be removed in *reader* 2.0); use `new` instead.

update_feeds_iter(*new_only=no value*, *workers=1*, ***, *new=no value*)
Update all the feeds that have updates enabled.

Parameters

- **new_only** (*bool*) – Only update feeds that have never been updated. Defaults to False.

  Deprecated since version 1.19: Use `new` instead.

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

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 is always a *ParseError*, but other *ReaderError* subclasses may be yielded in the future.

Raises *StorageError* –

New in version 1.14.

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

Deprecated since version 1.19: The `new_only` argument (will be removed in *reader* 2.0); use `new` instead.

update_feed(*feed*)
Update a single feed.

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

Parameters **feed** (*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*** –
- ***StorageError*** –

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

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

get_entries(*, *feed=None*, *entry=None*, *read=None*, *important=None*, *has_enclosures=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. Currently, that means:

- by import date for entries published less than 7 days ago
- by published date otherwise (if an entry does not have *published*, *updated* is used)

This is to make sure newly imported entries appear at the top regardless of when the feed says they were published (sometimes, it lies by a day or two).

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

`'random'`

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

New in version 1.2.

Parameters

- **feed** (*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*) – Only return (un)important entries.
- **has_enclosures** (*bool or None*) – Only return entries that (don't) have enclosures.
- **feed_tags** (*None or bool or list(str or bool or list(str or bool))*) – Only return the entries from feeds matching these tags; works like the *get_feeds()* `tags` argument.
- **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 1.2: The `sort` keyword argument.

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

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

get_entry(*entry*, *default=no value*)

Get an entry.

Like `next(iter(reader.get_entries(entry=entry)), default)`, 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*** –

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

Count all or some of the entries.

See *get_entries()* for details on how filtering works.

Parameters

- **feed** (*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*) – Only count (un)important entries.
- **has_enclosures** (*bool or None*) – Only count entries that (don't) have enclosures.
- **feed_tags** (*None or bool or list(str or bool or list(str or bool))*) – Only count the entries from feeds matching these tags.

Returns

Return type *EntryCounts*

Raises ***StorageError*** –

New in version 1.11.

mark_entry_as_read(*entry*)
Mark an entry as read.

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

Raises

- *EntryNotFoundError* –
- *StorageError* –

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

mark_entry_as_unread(*entry*)
Mark an entry as unread.

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

Raises

- *EntryNotFoundError* –
- *StorageError* –

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

mark_entry_as_important(*entry*)
Mark an entry as important.

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

Raises

- *EntryNotFoundError* –
- *StorageError* –

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

mark_entry_as_unimportant(*entry*)
Mark an entry as unimportant.

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

Raises

- *EntryNotFoundError* –
- *StorageError* –

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

mark_as_read(*entry*)
Deprecated alias for *mark_entry_as_read()*.

Deprecated since version 1.18: This method will be removed in *reader* 2.0. Use *mark_entry_as_read()* instead.

mark_as_unread(*entry*)
Deprecated alias for *mark_entry_as_unread()*.

Deprecated since version 1.18: This method will be removed in *reader* 2.0. Use *mark_entry_as_unread()* instead.

mark_as_important(*entry*)
Deprecated alias for *mark_entry_as_important()*.

Deprecated since version 1.18: This method will be removed in *reader* 2.0. Use *mark_entry_as_important()* instead.

mark_as_unimportant(*entry*)
Deprecated alias for *mark_entry_as_unimportant()*.

Deprecated since version 1.18: This method will be removed in *reader* 2.0. Use *mark_entry_as_unimportant()* instead.

get_feed_metadata(*feed*, **args*, *key=None*)
Get all or some of the metadata for a feed as `(key, value)` pairs.

Parameters

- **feed** (*str or Feed*) – The feed URL.
- **key** (*str or None*) – Only return the metadata for this key.

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

Raises ***StorageError*** –

Changed in version 1.18: *iter_feed_metadata()* was renamed to *get_feed_metadata()*, and *get_feed_metadata()* was renamed to *get_feed_metadata_item()*.

To preserve backwards compatibility, the `get_feed_metadata(feed, key[, default]) -> value` form (positional arguments only) will continue to work as an alias for `get_feed_metadata_item(feed, key[, default])` until the last 1.* *reader* version, after which it will result in a `TypeError`.

get_feed_metadata_item(*feed*, *key*, *default=no value*)
Get metadata for a feed.

Like `next(iter(reader.get_feed_metadata(feed, key=key)), (None, default))[1]`, but raises a custom exception instead of `StopIteration`.

Parameters

- **feed** (*str or Feed*) – The feed URL.
- **key** (*str*) – The key of the metadata to retrieve.
- **default** – Returned if given and no metadata exists for *key*.

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

Return type JSONType

Raises

- ***FeedMetadataNotFoundError*** –
- ***StorageError*** –

New in version 1.18: Renamed from *get_feed_metadata()*.

set_feed_metadata_item(*feed*, *key*, *value*)
Set metadata for a feed.

Parameters

- **feed** (*str or Feed*) – The feed URL.
- **key** (*str*) – The key of the metadata item to set.

- **value** (*JSONType*) – The value of the metadata item to set. JSONType is whatever `json.dumps()` accepts.

Raises

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

New in version 1.18: Renamed from *set_feed_metadata()*.

delete_feed_metadata_item(*feed*, *key*)
Delete metadata for a feed.

Parameters

- **feed** (*str or Feed*) – The feed URL.
- **key** (*str*) – The key of the metadata item to delete.

Raises

- ***FeedMetadataNotFoundError*** –
- ***StorageError*** –

New in version 1.18: Renamed from *delete_feed_metadata()*.

iter_feed_metadata(*feed*, **args*, *key=None*)
Deprecated alias for *get_feed_metadata()*.

Deprecated since version 1.18: This method will be removed in *reader* 2.0. Use *get_feed_metadata()* instead.

set_feed_metadata(*feed*, *key*, *value*)
Deprecated alias for *set_feed_metadata_item()*.

Deprecated since version 1.18: This method will be removed in *reader* 2.0. Use *set_feed_metadata_item()* instead.

delete_feed_metadata(*feed*, *key*)
Deprecated alias for *delete_feed_metadata_item()*.

Deprecated since version 1.18: This method will be removed in *reader* 2.0. Use *delete_feed_metadata_item()* instead.

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.

Raises

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

search_entries(*query*, ***, *feed=None*, *entry=None*, *read=None*, *important=None*, *has_enclosures=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.

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 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*) – Only search (un)important entries.
- **has_enclosures** (*bool or None*) – Only search entries that (don't) have enclosures.
- **feed_tags** (*None or bool or list(str or bool or list(str or bool))*) – Only return the entries from feeds matching these tags; works like the *get_feeds()* `tags` argument.
- **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 1.4: The `sort` keyword argument.

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

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

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

Count entries matching a full-text search query.

See *search_entries()* for details on how the query syntax and filtering work.

Search must be enabled to call this method.

Parameters

- **query** (*str*) – The search query.
- **feed** (*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*) – Only count (un)important entries.
- **has_enclosures** (*bool or None*) – Only count entries that (don't) have enclosures.

- **feed_tags** (*None or bool or list(str or bool or list(str or bool))*) – Only count the entries from feeds matching these tags.

Returns

Return type *EntrySearchCounts*

Raises

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

New in version 1.11.

add_feed_tag(*feed*, *tag*)

Add a tag to a feed.

Adding a tag that the feed already has is a no-op.

Parameters

- **feed** (*str or Feed*) – The feed URL.
- **tag** (*str*) – The tag to add.

Raises

- *FeedNotFoundError* –
- *StorageError* –

New in version 1.7.

remove_feed_tag(*feed*, *tag*)

Remove a tag from a feed.

Removing a tag that the feed does not have is a no-op.

Parameters

- **feed** (*str or Feed*) – The feed URL.
- **tag** (*str*) – The tag to remove.

Raises *StorageError* –

New in version 1.7.

get_feed_tags(*feed=None*)

Get all or some of the feed tags.

Parameters **feed** (*str or Feed or None*) – Only return the tags for this feed.

Yields *str* – The tags, in alphabetical order.

Raises *StorageError* –

New in version 1.7.

make_reader_reserved_name(*key*)

Create a *reader*-reserved tag or metadata 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

New in version 1.17.

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

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

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

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

New in version 1.17.

property reserved_name_scheme

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)

4.1.2 Data objects

class reader.**Feed**(*url*, *updated=None*, *title=None*, *link=None*, *author=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-naive, and always represent UTC.

url
The URL of the feed.

updated = None
The date the feed was last updated, according to the feed.

title = None
The title of the feed.

link = None
The URL of a page associated with the feed.

author = None
The author of the feed.

user_title = None
User-defined feed title.

added = None
The date when the feed was added.

New in version 1.3.

last_updated = None
The date when the feed was last retrieved by reader.

New in version 1.3.

last_exception = None
If a *ParseError* happend during the last update, its cause.

New in version 1.3.

updates_enabled = True
Whether updates are enabled for this feed.

New in version 1.11.

property object_id
Alias for *url*.

New in version 1.12.

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

Data type representing information about an exception.

New in version 1.3.

type_name
The fully qualified name of the exception type.

value_str
String representation of the exception value.

traceback_str
String representation of the exception traceback.

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

Data type representing an entry.

All `datetime` attributes are timezone-naive, and always represent UTC.

property feed_url
The feed URL.

id
The entry id.

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

title = None
The title of the entry.

link = None
The URL of a page associated with the entry.

author = None
The author of the feed.

published = None
The date the entry was first published.

summary = None
A summary of the entry.

content = ()
Full content of the entry. A sequence of *Content* objects.

enclosures = ()
External files associated with the entry. A sequence of *Enclosure* objects.

read = False
Whether the entry was read or not.

important = False
Whether the entry is important or not.

last_updated = None
The date when the entry was last updated by reader.

New in version 1.3.

original_feed_url = 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 = None
The entry's feed.

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

New in version 1.12.

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

Data type representing a piece of content.

value
The content value.

type = None
The content type.

language = None
The content language.

class reader.**Enclosure**(*href*, *type=None*, *length=None*)
Data type representing an external file.

href
The file URL.

type = None
The file content type.

length = None
The file length.

class reader.**EntrySearchResult**(*feed_url*, *id*, *metadata=mappingproxy({})*, *content=mappingproxy({})*)
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
The feed URL.

id
The entry id.

metadata = mappingproxy({})
Matching entry metadata, in arbitrary order. Currently entry.title and entry.feed.user_title/.title.

content = mappingproxy({})
Matching entry content, sorted by relevance. Any of entry.summary and entry.content[].value.

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

New in version 1.12.

class reader.**HighlightedString**(*value=''*, *highlights=()*)
A string that has some of its parts highlighted.

value = ''
The underlying string.

highlights = ()
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 = None

Total number of feeds.

broken = None

Number of broken feeds.

updates_enabled = None

Number of feeds that have updates enabled.

class reader.**EntryCounts**(*total=None*, *read=None*, *important=None*, *has_enclosures=None*)
Count information about entries.

New in version 1.11.

total = None
Total number of entries.

read = None
Number of read entries.

important = None
Number of important entries.

has_enclosures = None
Number of entries that have enclosures.

class reader.**EntrySearchCounts**(*total=None*, *read=None*, *important=None*, *has_enclosures=None*)
Count information about entry search results.

New in version 1.11.

total = None
Total number of entries.

read = None
Number of read entries.

important = None
Number of important entries.

has_enclosures = None
Number of entries that have enclosures.

class reader.**UpdateResult**(*url*, *value*)
Named tuple representing the result of a feed update.

New in version 1.14.

property url
The URL of the feed.

property value
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.

ReaderError

If there was an error while updating the feed.

class reader.**UpdatedFeed**(*url*, *new*, *modified*)
The result of a successful feed update.

New in version 1.14.

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

url
The URL of the feed.

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

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

property updated
Deprecated alias for *UpdatedFeed.modified*.

class reader.**EntryUpdateStatus**(*value*)
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=''*)
A feed error occured.

Subclass of *ReaderError*.

url
The feed URL.

property object_id
Alias for *url*.

New in version 1.12.

exception reader.**FeedExistsError**(*url*, *message=''*)
Feed already exists.

Subclass of *FeedError*.

exception reader.**FeedNotFoundError**(*url*, *message=''*)
Feed not found.

Subclass of *FeedError*.

exception reader.**ParseError**(*url*, *message=''*)
An error occured while getting/parsing feed.

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

Subclass of *FeedError*.

exception reader.**EntryError**(*feed_url*, *id*, *message=''*)
An entry error occurred.

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

Subclass of *ReaderError*.

feed_url
The feed URL.

id
The entry id.

property object_id
Alias for (*url*, *id*).

New in version 1.12.

property url
Deprecated alias for *EntryError.feed_url*.

exception reader.**EntryNotFoundError**(*feed_url*, *id*, *message=''*)
Entry not found.

Subclass of *EntryError*.

exception reader.**MetadataError**(**args*, *key*, ***kwargs*)
A metadata error occurred.

Changed in version 1.18: Signature changed from `MetadataError(message='')` to `MetadataError(key, message='')`.

Subclass of *ReaderError*.

key
The metadata key.

exception reader.**MetadataNotFoundError**(**args*, *key*, ***kwargs*)
Metadata not found.

Changed in version 1.18: Signature changed from `MetadataNotFoundError(url, key, message='')` to `MetadataNotFoundError(key, message='')`.

Subclass of *MetadataError*.

exception reader.**FeedMetadataNotFoundError**(*url*, *key*, *message=''*)
Feed metadata not found.

New in version 1.18.

Subclass of *MetadataNotFoundError* and *FeedError*.

exception reader.**StorageError**(*message=''*)
An exception was raised by the underlying storage.

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

Subclass of *ReaderError*.

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

Subclass of *ReaderError*.

exception reader.**SearchNotEnabledError**(*message=''*)
A search-related method was called when search was not enabled.

Subclass of *SearchError*.

exception reader.InvalidSearchQueryError(*message=''*)
The search query provided was somehow invalid.

Subclass of *SearchError* and ValueError.

exception reader.PluginError(*message=''*)
A plugin-related exception.

Subclass of *ReaderError*.

exception reader.InvalidPluginError(*message=''*)
An invalid plugin was provided.

Subclass of *PluginError* and ValueError.

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

This part of the documentation covers the *reader* command-line interface.

> **Warning:** The CLI is not stable yet and might change without any notice.

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.

--config <config>
: Path to the reader config.

  Default /home/docs/.config/reader/config.yaml

--version
: Show the version and exit.

Environment variables

READER_DB
: Provide a default for *--db*

READER_PLUGIN
: Provide a default for *--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 stable yet and might change without any notice.

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
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

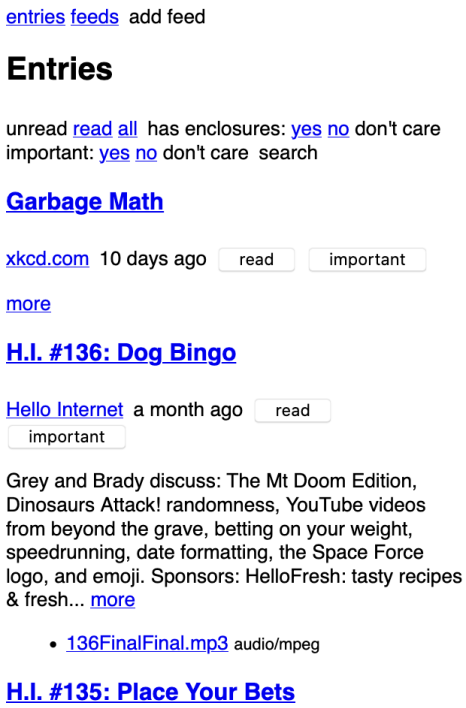

Fig. 1: main page

Feed page

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*.

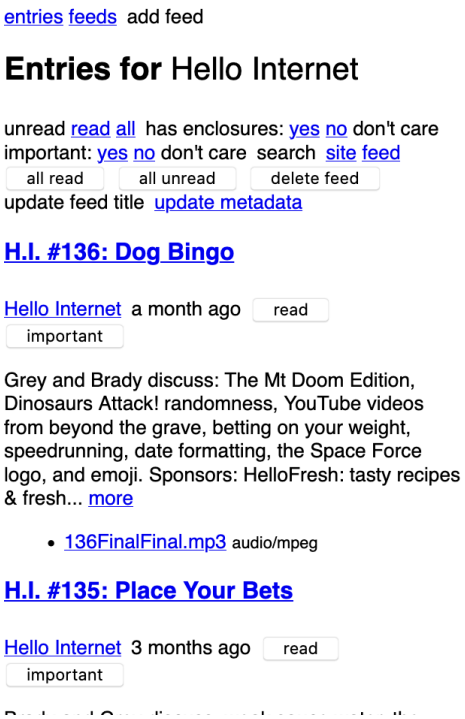

Fig. 2: feed page

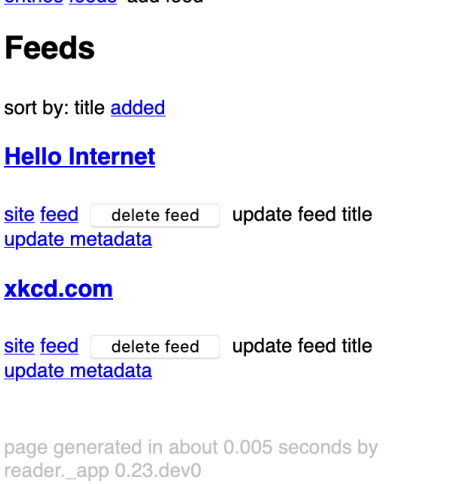

Fig. 3: feeds page

entries feeds add feed

Entry: Garbage Math

in xkcd.com 10 days ago read important

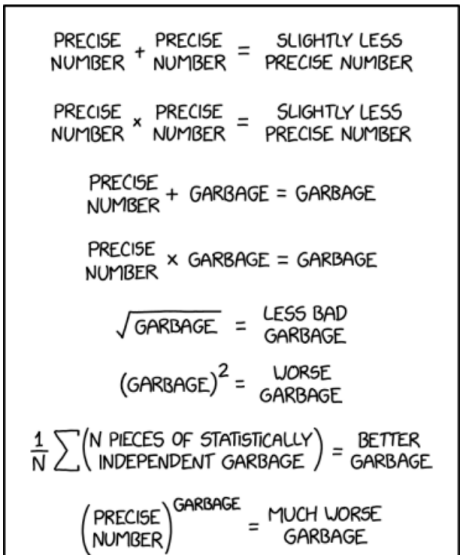

Fig. 4: entry page

entries feeds add feed

Entry: H.I. #136: Dog Bingo

by Hello Internet in Hello Internet a month ago
read important

Grey and Brady discuss: The Mt Doom Edition, Dinosaurs Attack! randomness, YouTube videos from beyond the grave, betting on your weight, speedrunning, date formatting, the Space Force logo, and emoji.

Sponsors:

HelloFresh: tasty recipes & fresh ingredients delivered to your door - get ten free meals including shipping - go to hellofresh.com/hellointernet10 and use promo code hellointernet10

Audible: the largest selection of audiobooks and original audio performances anywhere - start a 30-day trial and get 1 audiobook and 2 Audible Originals absolutely free by signing up at audible.com/hellointernet or text "hellointernet" to 500-500

Dashlane: password manager app and secure digital wallet - try Dashlane here for a free 30 day

Fig. 5: entry page

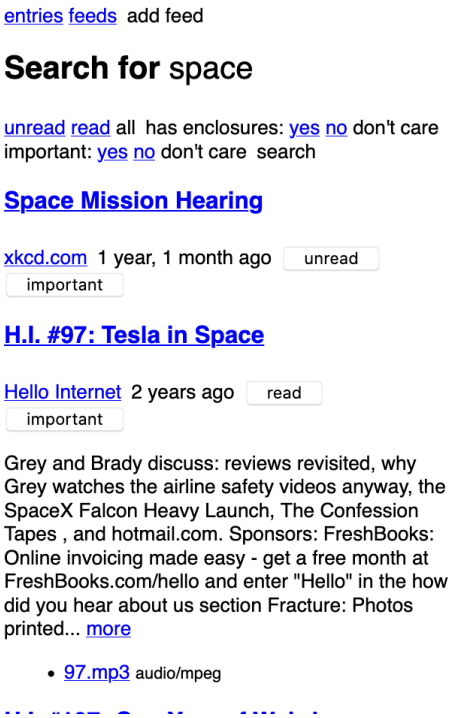

Fig. 6: search page

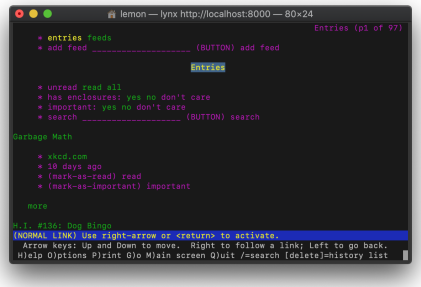

Fig. 7: Lynx

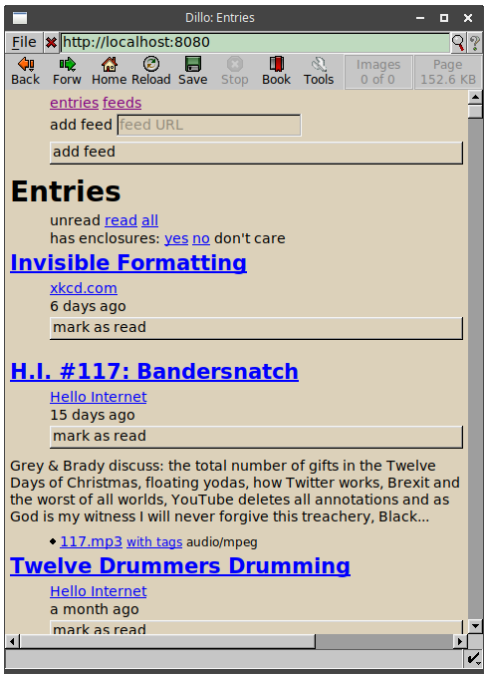

Fig. 8: Dillo

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).

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:
#
```

(continues on next page)

(continued from previous page)

```
#   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.tumblr_gdpr:tumblr_gdpr:
            reader.ua_fallback:


# CLI context.

cli:
    # 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:
```

(continues on next page)

(continued from previous page)

```
            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:
        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`. Because the entry id is used to uniquely identify entries, normally this results in the entry being added again with the new id.

This plugin addresses this by copying entry user attributes like *read* or *important* from the old entry to the new one.

Note: There are plans to *delete* the old entry after copying user attributes; please +1 / comment in #140 if you need this.

Duplicates are entries with the same title *and* the same summary/content, after all HTML tags and whitespace have been stripped.

Entry user attributes are set as follows:

`read`

If the old entry is read, the new one will be too. If the old entry is unread, it will be marked as read in favor of the new one.

before	after	
old.read	old.read	new.read
True	True	True
False	True	False

`important`

If the old entry is important, it will be marked as unimporant, and the new one will be marked as important.

before	after	
old.important	old.important	new.important
True	False	True
False	False	False

reader.mark_as_read

Mark added entries of specific feeds as read if their title matches a regex.

To configure, set the `make_reader_reserved_name('mark_as_read')` (by default, `.reader.mark_as_read`) feed metadata to something like:

```
{
    "title": ["first-regex", "second-regex"]
}
```

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 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
```

To load web application plugins, set the `READER_APP_PLUGIN` environment variable in a similar way.

For *built-in plugins*, it is enough to use the plugin name (`reader.XYZ`).

Note: `make_reader()` ignores the plugin environment variables.

5.4.3 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])
```

tumblr_gdpr

Accept Tumblr GDPR stuff.

Since May 2018, Tumblr redirects all new sessions to an "accept the terms of service" page, including RSS feeds (supposed to be machine-readable), breaking them.

This plugin "accepts the terms of service" on your behalf.

To load:

```
READER_PLUGIN='reader._plugins.tumblr_gdpr:tumblr_gdpr' \
python -m reader update -v
```

Implemented for https://github.com/lemon24/reader/issues/67.

Note: This plugin does not seem to be needed anymore as of August 2020.

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.

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.

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 serve
```

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 3.7+ and PyPy.

6.1 Backwards compatibility

reader uses semantic versioning.

This means you should never be afraid to upgrade *reader* between minor versions if you're using its public API.

If breaking compatibility will ever be needed, it will be 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 published.

That said, new major versions will be released as conservatively as possible. Even during the initial development phase (versions 0.*), over 20+ minor versions spanning 1.5 years, backwards compatibility was only broken 3 times, with the approriate deprecation warnings.

6.1.1 What is the public API

The *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 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).

While argument and attribute types are part of the public API, type annotations and type aliases (even if not private), are **not** part of the public API.

Other exceptions are possible; they will be marked aggresively as such.

> **Warning:** As of version 1.20, the *command-line interface*, *web application*, and *plugin system/hooks* are **not** part of the public API; they are not stable yet and might change without any notice.

6.2 Development

6.2.1 Goals

Goals:

- clearly documented API
- minimal web interface
- minimal CLI

Development should follow a problem-solution approach.

6.2.2 Roadmap

In no particular order:

- API to delete old entries. (#96)
- API to delete duplicate entries. (#140)
- Batch get related resources API. (#191)
- update_feeds() filtering. (#193)
- Web application re-design.
- Plugin system / hooks stabilization. (#80)
- Internal API stabilization.
- CLI stabilization.
- Web application stabilization.
- OPML support. (#165)

6.2.3 Style guide

reader uses the Black style.

You should enforce it by using pre-commit. To install it into your git hooks, run:

```
pip install pre-commit  # ./run.sh install-dev already does both
pre-commit install
```

Every time you clone the repo, running `pre-commit install` should always be the first thing you do.

6.2.4 Testing

First, install the testing dependencies:

```
./run.sh install-dev    # or
pip install '.[search,cli,app,tests,dev,unstable-plugins]'
```

Run tests using the current Python interpreter:

```
pytest --runslow
```

Run tests using the current Python interpreter, but skip slow tests:

```
pytest
```

Run tests for all supported Python versions:

```
tox
```

Run tests with coverage and generate an HTML report (in `./htmlcov`):

```
./run.sh coverage-all
```

Run the type checker:

```
./run.sh typing         # or
mypy --strict src
```

Start a local development server for the web application:

```
./run.sh serve-dev      # or

FLASK_DEBUG=1 FLASK_TRAP_BAD_REQUEST_ERRORS=1 \
FLASK_APP=src/reader/_app/wsgi.py \
READER_DB=db.sqlite flask run -h 0.0.0.0 -p 8000
```

6.2.5 Building the documentation

First, install the dependencie:

```
pip install '.[docs]'   # ./run.sh install-dev already does it for you
```

The documentation is built with Sphinx:

```
./run.sh docs           # or
make -C docs html       # using Sphinx's Makefile directly
```

The built HTML docs should be in `./docs/_build/html/`.

6.2.6 Making a release

Making a release (from `x` to `y == x + 1`):

Note: scripts/release.py already does most of these.

- (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 prompts) tag 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) deactivate old versions in Read the Docs

6.2.7 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.

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 (not added, as of 1.13): #155#issuecomment-673694472

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

Metrics

Some thoughts on implementing metrics: #68#issuecomment-450025175.

Query builder

Survey of possible options: #123#issuecomment-582307504.

Pagination for methods that return iterators

Why do it for the private implementation: #167#issuecomment-626753299 (also a comment in storage code).

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

reader types to Atom mapping

This whole issue: #153.

Sort by random

Some thoughts in the initial issue: #105.

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

Feed tags

Detailed requirements and API discussion: #184#issuecomment-689587006.

Entry user data

#228#issuecomment-810098748 discusses three different kinds, 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, can 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)

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.

Counts API

Detailed requirements and API discussion: #185#issuecomment-731743327.

Using None as a special argument value

This comment: #177#issuecomment-674786498.

Batch update (set) methods

There's a discussion on why I want to postpone this in this comment: #187#issuecomment-700740251.

Using a single Reader objects from multiple threads

Some thoughts on why it's difficult to do: #206#issuecomment-751383418.

Plugins

List of hooks (unmaintained as of 2021): #80.

Minimal plugin API (case study and considetrations for the built-in plugin naming scheme): *229#issuecomment-803870781*.

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.

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.

User interactions

Note: This list might lag behing 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.

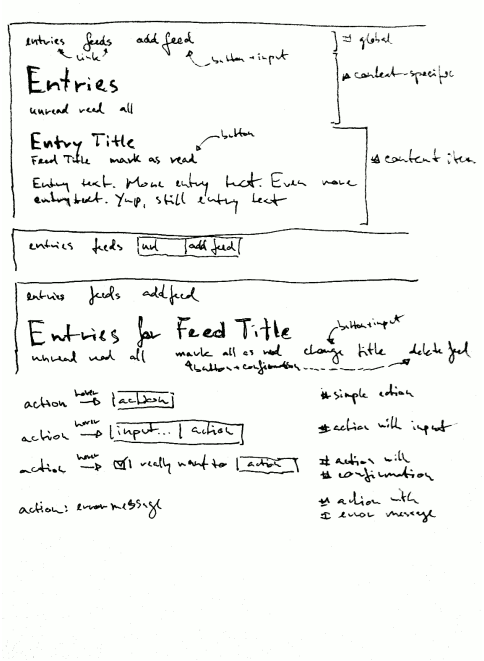

Fig. 1: page structure, controls

Pages

Text TBD.

6.3 Changelog

6.3.1 Version 1.20

Released 2021-07-12

- Add *Reader.after_entry_update_hooks*, which allows 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)

6.3.2 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*.

6.3.3 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)

6.3.4 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)

6.3.5 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.

6.3.6 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.

6.3.7 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)

6.3.8 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.

6.3.9 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)

6.3.10 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.

6.3.11 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)

6.3.12 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)

6.3.13 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)

6.3.14 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)

6.3.15 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)

6.3.16 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).

6.3.17 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.

6.3.18 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)

6.3.19 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)

6.3.20 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)

6.3.21 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)

6.3.22 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)

6.3.23 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)

6.3.24 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)

6.3.25 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)

6.3.26 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.

6.3.27 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)

6.3.28 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)

6.3.29 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)

6.3.30 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)

6.3.31 Version 0.14

Released 2019-08-12

- Drop Python 3.5 support. (#124)
- Improve entry ordering implementation. (#110)

6.3.32 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)

6.3.33 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.

6.3.34 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)

6.3.35 Version 0.10

Released 2019-05-18

- Unify plugin loading and error handling code. (#112)
- Minor improvements to CLI error reporting.

6.3.36 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)

6.3.37 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`).

6.3.38 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)

6.3.39 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)

6.3.40 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).

6.3.41 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)

6.3.42 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.

6.3.43 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.

6.3.44 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)

6.3.45 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

reader, 25
reader._plugins.enclosure_tags, 65
reader._plugins.preview_feed_list, 65
reader._plugins.sqlite_releases, 65
reader._plugins.tumblr_gdpr, 64
reader.plugins.enclosure_dedupe, 62
reader.plugins.entry_dedupe, 63
reader.plugins.mark_as_read, 63
reader.plugins.ua_fallback, 63

INDEX

Symbols

--config <config>
    reader command line option, 50
--db <db>
    reader command line option, 50
--host <host>
    reader-serve command line option, 54
--merge
    reader-config-dump command line option, 51
--new-only
    reader-update command line option, 54
--no-merge
    reader-config-dump command line option, 51
--no-new-only
    reader-update command line option, 54
--no-update
    reader-add command line option, 51
--plugin <plugin>
    reader command line option, 50
    reader-serve command line option, 54
--port <port>
    reader-serve command line option, 54
--update
    reader-add command line option, 51
--verbose
    reader-add command line option, 51
    reader-remove command line option, 52
    reader-search-update command line option, 53
    reader-serve command line option, 54
    reader-update command line option, 54
--version
    reader command line option, 50
--workers <workers>
    reader-update command line option, 54
-h
    reader-serve command line option, 54
-p
    reader-serve command line option, 54
-v
    reader-add command line option, 51
    reader-remove command line option, 52
    reader-search-update command line option, 53
    reader-serve command line option, 54
    reader-update command line option, 54

A

add_feed() (*reader.Reader method*), 27
add_feed_tag() (*reader.Reader method*), 39
added (*reader.Feed attribute*), 41
after_entry_update_hooks (*reader.Reader attribute*), 27
apply() (*reader.HighlightedString method*), 44
author (*reader.Entry attribute*), 42
author (*reader.Feed attribute*), 41

B

broken (*reader.FeedCounts attribute*), 44

C

change_feed_url() (*reader.Reader method*), 28
close() (*reader.Reader method*), 27
Content (*class in reader*), 42
content (*reader.Entry attribute*), 42
content (*reader.EntrySearchResult attribute*), 43

D

delete_feed() (*reader.Reader method*), 27
delete_feed_metadata() (*reader.Reader method*), 36
delete_feed_metadata_item() (*reader.Reader method*), 36
disable_feed_updates() (*reader.Reader method*), 30
disable_search() (*reader.Reader method*), 36

E

enable_feed_updates() (*reader.Reader method*), 30
enable_search() (*reader.Reader method*), 36
Enclosure (*class in reader*), 43
enclosures (*reader.Entry attribute*), 42
Entry (*class in reader*), 41
EntryCounts (*class in reader*), 44

EntryError, 46
EntryNotFoundError, 47
EntrySearchCounts (*class in reader*), 45
EntrySearchResult (*class in reader*), 43
EntryUpdateStatus (*class in reader*), 46
ExceptionInfo (*class in reader*), 41
extract() (*reader.HighlightedString class method*), 43

F

Feed (*class in reader*), 41
feed (*reader.Entry attribute*), 42
feed_url (*reader.Entry property*), 42
feed_url (*reader.EntryError attribute*), 46
feed_url (*reader.EntrySearchResult attribute*), 43
FeedCounts (*class in reader*), 44
FeedError, 46
FeedExistsError, 46
FeedMetadataNotFoundError, 47
FeedNotFoundError, 46

G

get_entries() (*reader.Reader method*), 32
get_entry() (*reader.Reader method*), 33
get_entry_counts() (*reader.Reader method*), 33
get_feed() (*reader.Reader method*), 29
get_feed_counts() (*reader.Reader method*), 29
get_feed_metadata() (*reader.Reader method*), 35
get_feed_metadata_item() (*reader.Reader method*), 35
get_feed_tags() (*reader.Reader method*), 39
get_feeds() (*reader.Reader method*), 28

H

has_enclosures (*reader.EntryCounts attribute*), 45
has_enclosures (*reader.EntrySearchCounts attribute*), 45
HighlightedString (*class in reader*), 43
highlights (*reader.HighlightedString attribute*), 43
href (*reader.Enclosure attribute*), 43

I

id (*reader.Entry attribute*), 42
id (*reader.EntryError attribute*), 47
id (*reader.EntrySearchResult attribute*), 43
important (*reader.Entry attribute*), 42
important (*reader.EntryCounts attribute*), 45
important (*reader.EntrySearchCounts attribute*), 45
InvalidPluginError, 48
InvalidSearchQueryError, 47
is_search_enabled() (*reader.Reader method*), 36
iter_feed_metadata() (*reader.Reader method*), 36

K

key (*reader.MetadataError attribute*), 47

L

language (*reader.Content attribute*), 43
last_exception (*reader.Feed attribute*), 41
last_updated (*reader.Entry attribute*), 42
last_updated (*reader.Feed attribute*), 41
length (*reader.Enclosure attribute*), 43
link (*reader.Entry attribute*), 42
link (*reader.Feed attribute*), 41

M

make_plugin_reserved_name() (*reader.Reader method*), 40
make_reader() (*in module reader*), 25
make_reader_reserved_name() (*reader.Reader method*), 39
mark_as_important() (*reader.Reader method*), 34
mark_as_read() (*reader.Reader method*), 34
mark_as_unimportant() (*reader.Reader method*), 35
mark_as_unread() (*reader.Reader method*), 34
mark_entry_as_important() (*reader.Reader method*), 34
mark_entry_as_read() (*reader.Reader method*), 34
mark_entry_as_unimportant() (*reader.Reader method*), 34
mark_entry_as_unread() (*reader.Reader method*), 34
metadata (*reader.EntrySearchResult attribute*), 43
MetadataError, 47
MetadataNotFoundError, 47
MODIFIED (*reader.EntryUpdateStatus attribute*), 46
modified (*reader.UpdatedFeed attribute*), 46
module
    reader, 25
    reader._plugins.enclosure_tags, 65
    reader._plugins.preview_feed_list, 65
    reader._plugins.sqlite_releases, 65
    reader._plugins.tumblr_gdpr, 64
    reader.plugins.enclosure_dedupe, 62
    reader.plugins.entry_dedupe, 63
    reader.plugins.mark_as_read, 63
    reader.plugins.ua_fallback, 63

N

NEW (*reader.EntryUpdateStatus attribute*), 46
new (*reader.UpdatedFeed attribute*), 45

O

object_id (*reader.Entry property*), 42
object_id (*reader.EntryError property*), 47
object_id (*reader.EntrySearchResult property*), 43
object_id (*reader.Feed property*), 41
object_id (*reader.FeedError property*), 46
original_feed_url (*reader.Entry attribute*), 42

P

ParseError, 46
PluginError, 48
published (*reader.Entry attribute*), 42

Q

QUERY
    reader-search-entries command line option, 53

R

read (*reader.Entry attribute*), 42
read (*reader.EntryCounts attribute*), 45
read (*reader.EntrySearchCounts attribute*), 45
reader
    module, 25
Reader (*class in reader*), 26
reader command line option
    --config <config>, 50
    --db <db>, 50
    --plugin <plugin>, 50
    --version, 50
reader._plugins.enclosure_tags
    module, 65
reader._plugins.preview_feed_list
    module, 65
reader._plugins.sqlite_releases
    module, 65
reader._plugins.tumblr_gdpr
    module, 64
reader.plugins.enclosure_dedupe
    module, 62
reader.plugins.entry_dedupe
    module, 63
reader.plugins.mark_as_read
    module, 63
reader.plugins.ua_fallback
    module, 63
reader-add command line option
    --no-update, 51
    --update, 51
    --verbose, 51
    -v, 51
    URL, 51
reader-config-dump command line option
    --merge, 51
    --no-merge, 51
reader-remove command line option
    --verbose, 52
    -v, 52
    URL, 52
reader-search-entries command line option
    QUERY, 53
reader-search-update command line option
    --verbose, 53
    -v, 53
reader-serve command line option
    --host <host>, 54
    --plugin <plugin>, 54
    --port <port>, 54
    --verbose, 54
    -h, 54
    -p, 54
    -v, 54
reader-update command line option
    --new-only, 54
    --no-new-only, 54
    --verbose, 54
    --workers <workers>, 54
    -v, 54
    URL, 55
ReaderError, 46
remove_feed() (*reader.Reader method*), 28
remove_feed_tag() (*reader.Reader method*), 39
reserved_name_scheme (*reader.Reader property*), 40

S

search_entries() (*reader.Reader method*), 37
search_entry_counts() (*reader.Reader method*), 38
SearchError, 47
SearchNotEnabledError, 47
set_feed_metadata() (*reader.Reader method*), 36
set_feed_metadata_item() (*reader.Reader method*), 35
set_feed_user_title() (*reader.Reader method*), 30
split() (*reader.HighlightedString method*), 44
StorageError, 47
summary (*reader.Entry attribute*), 42

T

title (*reader.Entry attribute*), 42
title (*reader.Feed attribute*), 41
total (*reader.EntryCounts attribute*), 45
total (*reader.EntrySearchCounts attribute*), 45
total (*reader.FeedCounts attribute*), 44
traceback_str (*reader.ExceptionInfo attribute*), 41
type (*reader.Content attribute*), 43
type (*reader.Enclosure attribute*), 43
type_name (*reader.ExceptionInfo attribute*), 41

U

update_feed() (*reader.Reader method*), 31
update_feeds() (*reader.Reader method*), 30
update_feeds_iter() (*reader.Reader method*), 31
update_search() (*reader.Reader method*), 37
updated (*reader.Entry attribute*), 42

updated (*reader.Feed attribute*), 41
updated (*reader.UpdatedFeed property*), 46
UpdatedFeed (*class in reader*), 45
UpdateResult (*class in reader*), 45
updates_enabled (*reader.Feed attribute*), 41
updates_enabled (*reader.FeedCounts attribute*), 44
URL
    reader-add command line option, 51
    reader-remove command line option, 52
    reader-update command line option, 55
url (*reader.EntryError property*), 47
url (*reader.Feed attribute*), 41
url (*reader.FeedError attribute*), 46
url (*reader.UpdatedFeed attribute*), 45
url (*reader.UpdateResult property*), 45
user_title (*reader.Feed attribute*), 41

V

value (*reader.Content attribute*), 42
value (*reader.HighlightedString attribute*), 43
value (*reader.UpdateResult property*), 45
value_str (*reader.ExceptionInfo attribute*), 41