isbnlib Documentation

Release 3.7.2

Alexandre Conde

Jun 19, 2017

Contents

1 Info **3**
- 1.1 Usage . . . 3
- 1.2 Projects using *isbnlib* . . . 4

2 Install **5**

3 For Devs **7**
- 3.1 Note . . . 7
- 3.2 API's Main Namespaces . . . 7
- 3.3 Plugins . . . 9
- 3.4 Merge Metadata . . . 10
- 3.5 A full featured app! . . . 10

4 Known Issues **11**

5 ISBN **13**

6 Code **15**
- 6.1 Search . . . 15
- 6.2 Status . . . 15
- 6.3 How to Contribute . . . 15

isbnlib provides several useful methods and functions to validate, clean, transform, hyphenate and get metadata for ISBN strings.

Contents:

CHAPTER 1

Info

`isbnlib` is a (pure) python library that provides several useful methods and functions to validate, clean, transform, hyphenate and get metadata for ISBN strings. Its origin was as the core of isbntools.

This short version, is suitable to be include as a dependency in other projects. Has a straightforward setup and a very easy programmatic api.

Runs on py26, py27, py33, py34, py35, py36, pypy and pypy3.

Usage

Typical usage (as library):

```
import isbnlib
...
```

Just for fun, suppose I want the *most spoken about* book with certain words in his title. For a *quick-and-dirty solution*, enter the following code in a file and save it as `isbn_tmsa_book.py`.

```
#!/usr/bin/env python
import sys
from isbnlib import *

query = sys.argv[1].replace(' ', '+')
isbn = isbn_from_words(query)

print("The ISBN of the most `spoken-about` book with this title is %s" % isbn)
print("")
print("... and the book is:")
print("")
print((meta(isbn)))
```

Then in a command line (in the same directory):

```
$ python isbn_tmsa_book.py 'noise'
```

In my case I get:

```
The ISBN of the most `spoken-about` book with this title is 9780143105985

... and the book is:

{'Publisher': u'Penguin Books', 'Language': u'eng', 'Title': u'White noise',
'Year': u'2009', 'ISBN-13': u'9780143105985', 'Authors': u'Don DeLillo ;
introduction by Richard Powers.'}
```

Have fun!

Projects using *isbnlib*

isbntools https://github.com/xlcnd/isbntools

Spreads https://github.com/DIYBookScanner/spreads

BiblioManager https://github.com/Phyks/BMC/

Alessandria https://gitlab.com/openlabmatera/alessandria

CHAPTER 2

Install

From the command line enter (in some cases you have to preced the command with `sudo`):

```
$ pip install isbnlib
```

or:

```
$ easy_install isbnlib
```

or:

```
$ pip install isbnlib-3.7.2.tar.gz
```

(first you have to download the file!)

If you use linux systems, you can install using your distribution package manager (all major distributions have packages `python-isbnlib` and `python3-isbnlib`), however (usually) are **very old and don't work well anymore**!

CHAPTER 3

For Devs

Note

The official form of an ISBN is something like `ISBN 979-10-90636-07-1`. However for most applications only the numbers are important and you can always masked them if you need (see below). This library works mainly with 'striped' ISBNs (only numbers and X) like '0826497527'. You can strip an ISBN's like string by using `canonical(isbnlike)`. You can 'mask' the ISBN by using `mask(isbn)`. So in the examples below, when you see 'isbn' in the argument, it is a 'striped' ISBN, when the argument is an 'isbnlike' it is a string like `ISBN 979-10-90636-07-1` or even something dirty like `asdf 979-10-90636-07-1 bla bla`.

Two important concepts: **valid ISBN** should be an ISBN that was built according with the rules, this is distinct from **issued ISBN** that is an ISBN that was already issued to a publisher (this is the usage of the libraries and most of the web services). However *isbn.org*, probably by legal reasons, merges the two! So, according to *isbn.org*, '9786610326266' is not valid (because the block 978-66... has not been issued yet, however if you use `is_isbn13('9786610326266')` you will get `True` (because '9786610326266' follows the rules of an ISBN). But the situation is even murkier, try `meta('9786610326266')` and you will see that this ISBN was already used!

If possible, work with ISBNs in the isbn-13 format (since 2007, only are issued ISBNs in the isbn-13 format). You can always convert isbn-10 to isbn-13, but **not** the reverse. Read more about ISBN at isbn-international.org.

API's Main Namespaces

In the namespace `isbnlib` you have access to the core methods:

`is_isbn10(isbn10like)` Validates as ISBN-10.

`is_isbn13(isbn13like)` Validates as ISBN-13.

`to_isbn10(isbn13)` Transforms an isbn-13 to isbn-10.

`to_isbn13(isbn10)` Transforms an isbn-10 to isbn-13.

canonical(isbnlike) Keeps only numbers and X. You will get strings like *9780321534965*.

clean(isbnlike) Cleans ISBN (only legal characters).

notisbn(isbnlike, level='strict') Check with the goal to invalidate isbn-like.

get_isbnlike(text, level='normal') Extracts all substrings that seem like ISBNs (very useful for scraping).

get_canonical_isbn(isbnlike, output='bouth') Extracts ISBNs and transform them to the canonical form.

EAN13(isbnlike) Transforms an *isbnlike* string into an EAN13 number (validated canonical ISBN-13).

info(isbn) Gets the language or country assigned to this ISBN.

mask(isbn, separator='-') *Mask* (hyphenate) a canonical ISBN.

meta(isbn, service='default', cache='default') Gives you the main metadata associated with the ISBN. As *service* parameter you can use: `'wcat'` uses **worldcat.org** (**no key is needed**), `'goob'` uses the **Google Books service** (**no key is needed**), `'isbndb'` uses the **isbndb.com** service (**an api key is needed**), `'openl'` uses the **OpenLibrary.org** api (**no key is needed**), `merge` uses a merged record of `wcat`, `goob` and `openl` records (**no key is needed**) and **is the default option**. You can get an API key for the *isbndb.com service* **here_**. You can enter API keys with `isbnlib.config.add_apikey(service, apikey)`. The output can be formatted as `bibtex`, `msword`, `endnote`, `refworks`, `opf` or `json` (BibJSON) bibliographic formats with `isbnlib.registry.bibformatters`. `cache` only allows two values: 'default' or None. You can change the kind of cache by using `isbnlib.registry.set_cache` (see below). Now, you can extend the functionality of this function by adding pluggins, more metadata providers or new bibliographic formatters (check for available pluggins).

editions(isbn, service='merge') Returns the list of ISBNs of editions related with this ISBN. By default uses 'merge' (merges 'openl' with 'thingl'), but other providers are available: 'openl' users **Open Library**, 'thingl' (uses the service ThingISBN from **LibraryThing**) and 'any' (first tries 'openl', if no data, tries 'thingl').

isbn_from_words(words) Returns the most probable ISBN from a list of words (for your geographic area).

goom(words) Returns a list of references from **Google Books multiple references**.

doi(isbn) Returns a DOI's ISBN-A from a ISBN-13.

doi2tex(DOI) Returns metadata formated as BibTeX for a given DOI.

ren(filename) Renames a file using metadata from an ISBN in his filename.

desc(isbn) Returns a small description of the book. *Almost all data available are for US books!*

cover(isbn) Returns a dictionary with the url for cover. *Almost all data available are for US books!*

See files test_core and test_ext for **a lot of examples**.

The exceptions raised by these methods can all be catched using `ISBNLibException`.

You can extend the lib by using the classes and functions exposed in namespace `isbnlib.dev`, namely:

- `WEBService` a class that handles the access to web services (just by passing an url) and supports `gzip`. You can subclass it to extend the functionality... but probably you don't need to use it! It is used in the next class.
- `WEBQuery` a class that uses `WEBService` to retrieve and parse data from a web service. You can build a new provider of metadata by subclassing this class. His main methods allow passing custom functions (*handlers*) that specialize them to specific needs (`data_checker` and `parser`). It implements a **throttling mechanism** with a default rate of one call per second per service.
- `Metadata` a class that structures, cleans and 'validates' records of metadata. His method `merge` allows to implement a simple merging procedure for records from different sources. The main features can be implemented by a call to `stdmeta` function!

- `vias` exposes several functions to put calls to services, just by passing the name and a pointer to the service's `query` function. `vias.parallel` allows to put threaded calls. You can use `vias.serial` to make serial calls and `vias.multi` to use several cores. The default is `vias.serial`, but you can change that in the conf file.

The exceptions raised by these methods can all be catched using `ISBNLibDevException`. You **should't raise** this exception in your code, only raise the specific exceptions exposed in `isbnlib.dev` whose name end in Error.

In `isbnlib.dev.helpers` you can find several methods, that we found very useful, some of then are only used in `isbntools` (*an app and framework* that uses `isbnlib`).

With `isbnlib.registry` you can change the metadata service to be used by default (`setdefaultservice`), add a new service (`add_service`), access bibliographic formatters for metadata (`bibformatters`), set the default formatter (`setdefaultbibformatter`), add new formatters (`add_bibformatter`) and set a new cache (`set_cache`) (e.g. to switch off the chache `set_cache(None)`). The cache only works for calls through `isbnlib.meta`. These changes only work for the 'current session', so should be done always before calling other methods.

Finally, from `isbnlib.config` you can read and set configuration options: change timeouts with `seturlopentimeout` and `setthreadstimeout`, access api keys with `apikeys` and add new one with `add_apikey` and access and set generic and user-defined options with `options` and `set_option`.

Let us concretize the last point with a small example.

Suppose you want a small script to get metadata using `isbndb.org` formated in BibTeX.

To use this service you need an api-key (get it **here_**). A minimal script would be:

```
from isbnlib import meta
from isbnlib.config import add_apikey
from isbnlib.registry import bibformatters

SERVICE = 'isbndb'
APIKEY = 'THiSIsfAKe'  # <-- replace with YOUR key

# register your key
add_apikey(SERVICE, APIKEY)

# now you can use the service
isbn = '9780446310789'
bibtex = bibformatters['bibtex']
print(bibtex(meta(isbn, SERVICE)))
```

All these classes follow a simple design pattern and, if you follow it, will be very easy to integrate your classes with the rest of the lib.

Plugins

You can extend the functionality of the library by adding pluggins (for now, just new metadata providers or new bibliographic formatters).

Start with this template and follow the instructions there. For inspiration take a look at wcat, goob or merge.

After install, your pluggin will blend transparently in `isbnlib`.

Remember that plugins **must** support python 2.6+ and python 3.3+ (see python-future.org).

Merge Metadata

The original quality of metadata, at the several services, is not very good! If you need high quality metadata in your app, the only solution is to use *polling & merge* of several providers **and** a **lot** of cleaning and standardization for fields like `Authors` and `Publisher`. A *simple merge* provider is now the default in `isbnlib.meta`. It gives priority to `wcat` but overwrites the `Authors`, `Publisher` and `Year` fields with values from `goob` (if available) or `openl`. Uses the `merge` method of `Metadata` and *parallel* calls to services.

You can change that by using `vias`'s other methods (e.g. `isbnlib.config.set_option('VIAS_MERGE', 'multi')`.

You can write your own *merging scheme* by creating a new provider (see `merge` for an example).

Note: These classes are optimized for one-calls to services and not for batch calls.

A full featured app!

If you want a full featured app, that uses `isbnlib`, with end user apps, configuration files and a framework to further developement, take a look at isbntools.

You can browse the code, in a very structured way, at sourcegraph or GitHub.

CHAPTER 4

Known Issues

1. The `meta` method sometimes give a wrong result (this is due to errors on the chosen service), in alternative you could try one of the others services.
2. The `isbnlib` works internally with unicode, however this doesn't solve errors of lost information due to bad encode/decode at the origin!
3. Periodically, agencies, issue new blocks of ISBNs. The range of these blocks is on a database that `mask` uses. So it could happen, if you have a version of `isbnlib` that is too old, `mask` doesn't work for valid (recent) issued ISBNs. The solution? **Update isbnlib often**!
4. Calls to metadata services are cached by default. You can change that by calling `isbnlib.meta(isbn, service='default', cache=None)`

Any issue that you would like to report, please do it at github or post a question on stackoverflow (with tag **isbnlib**).

CHAPTER 5

ISBN

To know about ISBN:

- http://en.wikipedia.org/wiki/International_Standard_Book_Number
- http://www.isbn-international.org/

CHAPTER 6

Code

Search

Search/Browse the code at sourcegraph or github

Status

How to Contribute

`isbnlib` has a very small code base, so it is a good project to begin your adventure in open-source...

Main Steps

1. Make sure you have a GitHub account
2. Submit a ticket for your issue or idea, on GitHub issues (if possible wait for some feedback before any serious commitment... :)
3. Fork the repository on GitHub
4. `pip install -r requirements-dev.txt`
5. Do your code... (**remember the code must run on python 2.6, 2.7, 3.3, 3.4, pypy and be OS independent**) (you will find `travis-ci.org` very handy for this!)
6. Write tests for your code using `nose` and put then in the directory `isbnlib/test`
7. Pass **all tests** and with **coverage > 90%**. Check the coverage in Coveralls.
8. **Check if all requirements are fulfilled**!

9. Make a pull request on github...

Important

If you don't have experience in these issues, don't be put off by these requirements, see them as a learning opportunity. Thanks!

> For full instructions read the CONTRIBUTING doc.