EbookLib Documentation

Release 0.17

Aleksandar Erkalovic

Aug 11, 2018

Contents

EbookLib is a Python library for managing EPUB2/EPUB3 files. It's capable of reading and writing EPUB files programmatically.

The API is designed to be as simple as possible, while at the same time making complex things possible too. It has support for covers, table of contents, spine, guide, metadata, SMIL, pagebreaks and more. EbookLib works with Python 2.7 and Python 3.3.

Homepage: https://github.com/aerkalov/ebooklib/

CHAPTER 1

Tutorial

1.1 Introduction

Ebooklib is used to manage EPUB2/EPUB3 files.

1.2 Reading EPUB

```python
import ebooklib
from ebooklib import epub

book = epub.read_epub('test.epub')
```

There is a *ebooklib.epub.read_epub()* function used for reading EPUB files. It accepts full file path to the EPUB file as argument. It will return instance of *ebooklib.epub.EpubBook* class.

1.2.1 Metadata

Method *ebooklib.epub.EpubBook.get_metadata()* is used for fetching metadata. It accepts 2 arguments. First argument is name of the namespace ('DC' for Dublin Core and 'OPF' for custom metadata). Second argument is name of the key. It always returns a list. List will be empty if nothing is defined for that key.

Minimal required metadata from Dublic Core set (for EPUB3) is:

- DC:identifier
- DC:title
- DC:language

'DC' namespace is used when accessing Dublin Core metadata.

```
>>> book.get_metadata('DC', 'title')
[('Ratio', {})]

>>> book.get_metadata('DC', 'creator')
[('Firstname Lastname ', {})]

>>> book.get_metadata('DC', 'identifier')
[('9781416566120', {'id': 'isbn_9781416566120'})]
```

Optional metadata from the Dublic Core set is:

- DC:creator
- DC:contributor
- DC:publisher
- DC:rights
- DC:coverage
- DC:date
- DC:description

This is how Dublin Core metadata is defined inside of content.opf file.

```
<dc:language>en</dc:language>
<dc:identifier id="isbn_9781416566120">9781416566120</dc:identifier>
```

You can also have custom metadata. For instance this is how custom metadata is defined in content.opf file. You can define same key more then once. 2

```
<meta content="my-cover-image" name="cover"/>
<meta content="cover-image" name="cover"/>
```

When accessing custom metadata you will use namespace 'OPF'. Notice you will get more then one result now.

```
>>> book.get_metadata('OPF', 'cover')
[(None, {'content': 'my-cover-image', 'name': 'cover'}),
 (None, {'content': 'cover-image', 'name': 'cover'})]
```

Check the official documentation for more info:

- http://www.idpf.org/epub/30/spec/epub30-publications.html#sec-opf-dcmes-optional.
- http://www.idpf.org/epub/30/spec/epub30-publications.html
- http://dublincore.org/documents/dces/

1.2.2 Items

All of the resources (style sheets, images, videos, sounds, scripts and html files) are items.

```
images = book.get_items_of_type(ebooklib.ITEM_IMAGE)
```

Fetch items by their type with `ebooklib.epub.EpubBook.get_items_of_type()`.

Here is a list of current item types you can use:

- ITEM_UNKNOWN

- ITEM_IMAGE
- ITEM_STYLE
- ITEM_SCRIPT
- ITEM_NAVIGATION
- ITEM_VECTOR
- ITEM_FONT
- ITEM_VIDEO
- ITEM_AUDIO
- ITEM_DOCUMENT
- ITEM_COVER
- ITEM_SMIL

```
cover_image = book.get_item_with_id('cover-image')
```

Fetch items by their id (if you know it) with `ebooklib.epub.EpubBook.get_item_with_id()`.

```
index = book.get_item_with_href('index.xhtml')
```

Fetch them by their filename with `ebooklib.epub.EpubBook.get_item_with_href()`.

```
items = book.get_items_of_media_type('image/png')
```

Fetch them by their media type with `ebooklib.epub.EpubBook.get_items_of_type()`.

```
all_items = book.get_items()
```

Return all of the items with `ebooklib.epub.EpubBook.get_items()`. This is what you are going to use most of the times when handling unknown EPUB files.

Important to remember! Methods *get_item_with_id*, *get_item_with_href* will return item object. Methods *get_items_of_type*, *get_items_of_type* and *get_items* will return iterator (and not list).

To get a content from existing item (regarding if it is image, style sheet or html file) you use `ebooklib.epub.EpubItem.get_content()`. For HTML items you also have `ebooklib.epub.EpubHtml.get_body_content()`. What is the difference? Get_content always return entire content of the file while get_body_content only returns whatever is in the <body> part of the HTML document.

```
for item in book.get_items():
    if item.get_type() == ebooklib.ITEM_DOCUMENT:
        print('==================================')
        print('NAME : ', item.get_name())
        print('----------------------------------')
        print(item.get_content())
        print('==================================')
```

1.3 Creating EPUB

```
from ebooklib import epub

book = epub.EpubBook()
```

EPUB has some minimal metadata requirements which you need to fulfil. You need to define unique identifier, title of the book and language used inside. When it comes to language code recommended best practice is to use a controlled vocabulary such as RFC 4646 - http://www.ietf.org/rfc/rfc4646.txt.

```
book.set_identifier('sample123456')
book.set_title('Sample book')
book.set_language('en')

book.add_author('Aleksandar Erkalovic')
```

You can also add custom metadata. First one is from the Dublic Core namespace and second one is purely custom.

```
book.add_metadata('DC', 'description', 'This is description for my book')
book.add_metadata(None, 'meta', '', {'name': 'key', 'content': 'value'})
```

This is how our custom metadata will end up in the *content.opf* file.

```
<dc:description>This is description for my book</dc:description>
<meta content="value" name="key"></meta>
```

Chapters are represented by *ebooklib.epub.EpubHtml*. You must define the *file_name* and *title*. In our case title is going to be used when generating Table of Contents.

When defining content you can define it as valid HTML file or just parts of HTML elements you have as a content. It will ignore whatever you have in <head> element.

```
# intro chapter
c1 = epub.EpubHtml(title='Introduction',
                   file_name='intro.xhtml',
                   lang='en')
c1.set_content(u'<html><body><h1>Introduction</h1><p>Introduction paragraph.</p></
↪body></html>')

# about chapter
c2 = epub.EpubHtml(title='About this book',
                   file_name='about.xhtml')
c2.set_content('<h1>About this book</h1><p>This is a book.</p>')
```

Do some basic debugging to see what kind of content will end up in the book. In this case we have inserted title of the chapter and language definition. It would also add links to the style sheet files if we have attached them to this chapter.

```
>>> print(c1.get_content())
b'<?xml version=\'1.0\' encoding=\'utf-8\'?>\n<!DOCTYPE html>\n<html xmlns="http://
↪www.w3.org/1999/xhtml"
xmlns:epub="http://www.idpf.org/2007/ops" epub:prefix="z3998: http://www.daisy.org/
↪z3998/2012/vocab/structure/#"
lang="en" xml:lang="en">\n  <head>\n    <title>Introduction</title>\n  </head>\n
↪<body>\n    <h1>Introduction</h1>\n
<p>Introduction paragraph.</p>\n  </body>\n</html>\n'
```

Any kind of item (style sheet, image, HTML file) must be added to the book.

```
book.add_item(c1)
book.add_item(c2)
```

You can add any kind of file to the book. For instance, in this case we are adding style sheet file. We define filename, unique id, media_type and content for it. Just like the chapter files you need to add it to the book. Style sheet files could also be added to the chapter. In that case links would be automatically added to the chapter HTML.

```
style = 'body { font-family: Times, Times New Roman, serif; }'

nav_css = epub.EpubItem(uid="style_nav",
                        file_name="style/nav.css",
                        media_type="text/css",
                        content=style)
book.add_item(nav_css)
```

Table of the contents must be defined manually. ToC is a tuple/list of elements. You can either define link manually with *ebooklib.epub.Link* or just insert item object inside. When you manually insert you can define different title in the ToC than in the chapter. If you just insert item object it will use whatever title you defined for that item when creating it.

Sections are just tuple with two values. First one is title of the section and 2nd is tuple/list with subchapters.

```
book.toc = (epub.Link('intro.xhtml', 'Introduction', 'intro'),
              (
                epub.Section('Languages'),
                (c1, c2)
              )
            )
```

So as the Spine. You can use unique id for the item or just add instance of it to the spine.

```
book.spine = ['nav', c1, c2]
```

At the end we need to add NCX and Navigation tile. They will not be added automatically.

```
book.add_item(epub.EpubNcx())
book.add_item(epub.EpubNav())
```

At the end write down your book. You need to specify full path to the book, you can not write it down to the File Object or something else.

```
epub.write_epub('test.epub', book)
```

It also accepts some of the options.

Option	Default value
epub2_guide	True
epub3_landmark	True
epub3_pages	True
landmark_title	"Guide"
pages_title	"Pages"
spine_direction	True
package_direction	False
play_order	{'enabled': False, 'start_from': 1}

Example when overriding default options:

```
epub.write_epub('test.epub', book, {"epub3_pages": False})
```

1.4 Samples

Further examples are available in https://github.com/aerkalov/ebooklib/tree/master/samples

CHAPTER 2

Plugins

Plugins can be used to automate some of the workflow. For instance `ebooklib.plugins.tidyhtml` plugins automatically cleans HTML output for you. FootnotePlugin for instance rewrites custom footnotes into EPUB3 kind of footnotes. Without these plugins you would need to do wanted transformations before setting or getting content from your chapters.

There is a `ebooklib.plugins.base.BasePlugin` class which you need to extend. These are the methods you can override.

Method	Arguments	Description
before_write	book	Processing before book save
after_write	book	Processing after book save
before_read	book	Processing before book read
after_read	book	Processing after book read
item_after_read	book, item	Process general item after read
item_before_write	book, item	Process general item before write
html_after_read	book, chapter	Processing HTML before read
html_before_write	book, chapter	Processing HTML before save

2.1 Custom plugin

This is our use case. We have online WYSIWYG editing system for editing content of our books. In the editor to link to our other page we use this syntax *<a href=”../page/”>Our page</a>* but in the EPUB book we would need to link to *page.xhtml* file for instance.

We could do the transformations manually or we could write a plugin which does all of the work for us. We override *html_before_write* method. We parse the content of our chapter, find all links, replace them with new href and finally set new content to the chapter.

```
try:
    from urlparse import urlparse, urljoin
```

(continues on next page)

(continued from previous page)

```
except ImportError:
    from urllib.parse import urlparse, urljoin

from lxml import  etree

from ebooklib.plugins.base import BasePlugin
from ebooklib.utils import parse_html_string

class MyeLinks(BasePlugin):
    NAME = 'My Links'

    def html_before_write(self, book, chapter):
        try:
            tree = parse_html_string(chapter.content)
        except:
            return

        root = tree.getroottree()

        if len(root.find('body')) != 0:
            body = tree.find('body')

            for _link in body.xpath('//a'):
                _u = urlparse(_link.get('href', ''))

                # Let us care only for internal links at the moment
                if _u.scheme == '':
                    if _u.path != '':
                        _link.set('href', '%s.xhtml' % _u.path)

                    if _u.fragment != '':
                        _link.set('href', urljoin(_link.get('href'), '#%s' % _u.
↪fragment))

                    if _link.get('name') != None:
                        _link.set('id', _link.get('name'))
                        etree.strip_attributes(_link, 'name')

        chapter.content = etree.tostring(tree, pretty_print=True, encoding='utf-8')
```

When you want to use it just pass the list of plugins you want to use as extra option to the write_epub method (also to read_epub method). Plugins will be executed in the order they are defined in the list.

```
epub.write_epub('test.epub', book, {"plugins": [MyLinks()]})
```

CHAPTER 3

Frequently asked questions

3.1 How can I contribute?

Easily! We don't have any special procedure for contributions. Just send us bug reports, suggestions or Pull Requests!

- Find bugs and report them here - https://github.com/aerkalov/ebooklib/issues
- Have a suggestions or idea for a feature? Write it here - https://github.com/aerkalov/ebooklib/issues
- Clone and send us Pull Requests - https://github.com/aerkalov/ebooklib/

3.2 Read and save does not work

At the moment you can not use ebooklib to read EPUB file, write it down using ebooklib and expect to get file identical to the original. This is because ebooklib is opinionated when it comes to directory names, content it saves and etc. Different EPUB files will place style sheet files, images, fonts and html files to different directories than ebooklib. Ebooklib is also trying to produce valid EPUB3 output files, which is not always the case with all input files. A lot of times input files will not pass epubcheck validation.

Because of that we we suggest that you read EPUB file using ebooklib, create new book instance using ebooklib and transfer only things you care about. It is also up to you to change the paths for the links inside (to style sheets, images and other html files). When you are done just write it down.

3.3 What happened to the MOBI support?

We used to have branch with basic MOBI support. We stopped working on it as for our workflow KindleGen app (converting EPUB->MOBI) was more than enough and we never needed to parse MOBI files.

CHAPTER 4

ebooklib Package

4.1 `ebooklib` Package

4.2 `epub` Module

class ebooklib.epub.**EpubBook**

Bases: `object`

add_author(*author*, *file_as=None*, *role=None*, *uid='creator'*)

Add author for this document

add_item(*item*)

Add additional item to the book. If not defined, media type and chapter id will be defined for the item.

Args

- item: Item instance

add_metadata(*namespace*, *name*, *value*, *others=None*)

Add metadata

add_prefix(*name*, *uri*)

Appends custom prefix to be added to the content.opf document

```
>>> epub_book.add_prefix('bkterms', 'http://booktype.org/')
```

Args

- name: namespave name
- uri: URI for the namespace

get_item_with_href(*href*)

Returns item for defined HREF.

```
>>> book.get_item_with_href('EPUB/document.xhtml')
```

Args

- href: HREF for the item we are searching for

Returns Returns item object. Returns None if nothing was found.

get_item_with_id(*uid*)
Returns item for defined UID.

```
>>> book.get_item_with_id('image_001')
```

Args

- uid: UID for the item

Returns Returns item object. Returns None if nothing was found.

get_items()
Returns all items attached to this book.

Returns Returns all items as tuple.

get_items_of_media_type(*media_type*)
Returns all items of specified media type.

Args

- media_type: Media type for items we are searching for

Returns Returns found items as tuple.

get_items_of_type(*item_type*)
Returns all items of specified type.

```
>>> book.get_items_of_type(epub.ITEM_IMAGE)
```

Args

- item_type: Type for items we are searching for

Returns Returns found items as tuple.

get_metadata(*namespace*, *name*)
Retrieve metadata

get_template(*name*)
Returns value for the template.

Args

- name: template name

Returns Value of the template.

reset()
Initialises all needed variables to default values

set_cover(*file_name*, *content*, *create_page=True*)
Set cover and create cover document if needed.

Args

- file_name: file name of the cover page
- content: Content for the cover image
- create_page: Should cover page be defined. Defined as bool value (optional). Default value is True.

set_direction(*direction*)

Args

- direction: Options are "ltr", "rtl" and "default"

set_identifier(*uid*)
Sets unique id for this epub

Args

- uid: Value of unique identifier for this book

set_language(*lang*)
Set language for this epub. You can set multiple languages. Specific items in the book can have different language settings.

Args

- lang: Language code

set_template(*name*, *value*)
Defines templates which are used to generate certain types of pages. When defining new value for the template we have to use content of type 'str' (Python 2) or 'bytes' (Python 3).

At the moment we use these templates:

- ncx
- nav
- chapter
- cover

Args

- name: Name for the template
- value: Content for the template

set_title(*title*)
Set title. You can set multiple titles.

Args

- title: Title value

set_unique_metadata(*namespace*, *name*, *value*, *others=None*)
Add metadata if metadata with this identifier does not already exist, otherwise update existing metadata.

class ebooklib.epub.**EpubCover**(*uid='cover-img'*, *file_name=''*)
Bases: *ebooklib.epub.EpubItem*

Represents Cover image in the EPUB file.

get_type()
Guess type according to the file extension. Might not be the best way how to do it, but it works for now.

Items can be of type:

- ITEM_UNKNOWN = 0
- ITEM_IMAGE = 1
- ITEM_STYLE = 2
- ITEM_SCRIPT = 3
- ITEM_NAVIGATION = 4
- ITEM_VECTOR = 5
- ITEM_FONT = 6
- ITEM_VIDEO = 7
- ITEM_AUDIO = 8
- ITEM_DOCUMENT = 9
- ITEM_COVER = 10

We map type according to the extensions which are defined in ebooklib.EXTENSIONS.

Returns Returns type of the item as number.

class `ebooklib.epub.`**`EpubCoverHtml`**(*uid='cover'*, *file_name='cover.xhtml'*, *image_name=''*, *title='Cover'*)

Bases: *`ebooklib.epub.EpubHtml`*

Represents Cover page in the EPUB file.

`get_content`()

Returns content for cover page as HTML string. Content will be of type 'str' (Python 2) or 'bytes' (Python 3).

Returns Returns content of this document.

`is_chapter`()

Returns if this document is chapter or not.

Returns Returns book value.

exception `ebooklib.epub.`**`EpubException`**(*code*, *msg*)

Bases: `exceptions.Exception`

class `ebooklib.epub.`**`EpubHtml`**(*uid=None*, *file_name=''*, *media_type=''*, *content=None*, *title=''*, *lang=None*, *direction=None*, *media_overlay=None*, *media_duration=None*)

Bases: *`ebooklib.epub.EpubItem`*

Represents HTML document in the EPUB file.

`add_item`(*item*)

Add other item to this document. It will create additional links according to the item type.

Args

- item: item we want to add defined as instance of EpubItem

`add_link`(***kwgs*)

Add additional link to the document. Links will be embeded only inside of this document.

```
>>> add_link(href='styles.css', rel='stylesheet', type='text/css')
```

get_body_content()

Returns content of BODY element for this HTML document. Content will be of type 'str' (Python 2) or 'bytes' (Python 3).

Returns Returns content of this document.

get_content(*default=None*)

Returns content for this document as HTML string. Content will be of type 'str' (Python 2) or 'bytes' (Python 3).

Args

- default: Default value for the content if it is not defined.

Returns Returns content of this document.

get_language()

Get language code for this book item. Language of the book item can be different from the language settings defined globaly for book.

Returns As string returns language code.

get_links()

Returns list of additional links defined for this document.

Returns As tuple return list of links.

get_links_of_type(*link_type*)

Returns list of additional links of specific type.

Returns As tuple returns list of links.

get_type()

Always returns ebooklib.ITEM_DOCUMENT as type of this document.

Returns Always returns ebooklib.ITEM_DOCUMENT

is_chapter()

Returns if this document is chapter or not.

Returns Returns book value.

set_language(*lang*)

Sets language for this book item. By default it will use language of the book but it can be overwritten with this call.

class ebooklib.epub.**EpubImage**

Bases: *ebooklib.epub.EpubItem*

Represents Image in the EPUB file.

get_type()

Guess type according to the file extension. Might not be the best way how to do it, but it works for now.

Items can be of type:

- ITEM_UNKNOWN = 0
- ITEM_IMAGE = 1
- ITEM_STYLE = 2
- ITEM_SCRIPT = 3
- ITEM_NAVIGATION = 4
- ITEM_VECTOR = 5

- ITEM_FONT = 6
- ITEM_VIDEO = 7
- ITEM_AUDIO = 8
- ITEM_DOCUMENT = 9
- ITEM_COVER = 10

We map type according to the extensions which are defined in ebooklib.EXTENSIONS.

Returns Returns type of the item as number.

class `ebooklib.epub.`**`EpubItem`**(*uid=None*, *file_name=''*, *media_type=''*, *content=''*, *manifest=True*)

Bases: `object`

Base class for the items in a book.

`get_content`(*default=''*)

Returns content of the item. Content should be of type 'str' (Python 2) or 'bytes' (Python 3)

Args

- default: Default value for the content if it is not already defined.

Returns Returns content of the item.

`get_id`()

Returns unique identifier for this item.

Returns Returns uid number as string.

`get_name`()

Returns name for this item. By default it is always file name but it does not have to be.

Returns Returns file name for this item.

`get_type`()

Guess type according to the file extension. Might not be the best way how to do it, but it works for now.

Items can be of type:

- ITEM_UNKNOWN = 0
- ITEM_IMAGE = 1
- ITEM_STYLE = 2
- ITEM_SCRIPT = 3
- ITEM_NAVIGATION = 4
- ITEM_VECTOR = 5
- ITEM_FONT = 6
- ITEM_VIDEO = 7
- ITEM_AUDIO = 8
- ITEM_DOCUMENT = 9
- ITEM_COVER = 10

We map type according to the extensions which are defined in ebooklib.EXTENSIONS.

Returns Returns type of the item as number.

set_content(*content*)
Sets content value for this item.

Args

- content: Content value

class ebooklib.epub.**EpubNav**(*uid='nav'*, *file_name='nav.xhtml'*, *media_type='application/xhtml+xml'*)
Bases: `ebooklib.epub.EpubHtml`

Represents Navigation Document in the EPUB file.

is_chapter()
Returns if this document is chapter or not.

Returns Returns book value.

class ebooklib.epub.**EpubNcx**(*uid='ncx'*, *file_name='toc.ncx'*)
Bases: `ebooklib.epub.EpubItem`

Represents Navigation Control File (NCX) in the EPUB.

class ebooklib.epub.**EpubReader**(*epub_file_name*, *options=None*)
Bases: `object`

DEFAULT_OPTIONS = {}

load()

process()

read_file(*name*)

class ebooklib.epub.**EpubSMIL**(*uid=None*, *file_name="*, *content=None*)
Bases: `ebooklib.epub.EpubItem`

get_type()
Guess type according to the file extension. Might not be the best way how to do it, but it works for now.

Items can be of type:

- ITEM_UNKNOWN = 0
- ITEM_IMAGE = 1
- ITEM_STYLE = 2
- ITEM_SCRIPT = 3
- ITEM_NAVIGATION = 4
- ITEM_VECTOR = 5
- ITEM_FONT = 6
- ITEM_VIDEO = 7
- ITEM_AUDIO = 8
- ITEM_DOCUMENT = 9
- ITEM_COVER = 10

We map type according to the extensions which are defined in ebooklib.EXTENSIONS.

Returns Returns type of the item as number.

class `ebooklib.epub.`**`EpubWriter`**(*name*, *book*, *options=None*)

Bases: `object`

`DEFAULT_OPTIONS = {'epub2_guide': True, 'epub3_landmark': True, 'epub3_pages': True`

`process`()

`write`()

class `ebooklib.epub.`**`Link`**(*href*, *title*, *uid=None*)

Bases: `object`

class `ebooklib.epub.`**`Section`**(*title*, *href=''*)

Bases: `object`

`ebooklib.epub.`**`read_epub`**(*name*, *options=None*)

Creates new instance of EpubBook with the content defined in the input file.

```
>>> book = ebooklib.read_epub('book.epub')
```

Args

- name: full path to the input file
- options: extra options as dictionary (optional)

Returns Instance of EpubBook.

`ebooklib.epub.`**`write_epub`**(*name*, *book*, *options=None*)

Creates epub file with the content defined in EpubBook.

```
>>> ebooklib.write_epub('book.epub', book)
```

Args

- name: file name for the output file
- book: instance of EpubBook
- options: extra opions as dictionary (optional)

4.3 `utils` Module

`ebooklib.utils.`**`create_pagebreak`**(*pageref*, *label=None*, *html=True*)

`ebooklib.utils.`**`debug`**(*obj*)

`ebooklib.utils.`**`get_headers`**(*elem*)

`ebooklib.utils.`**`get_pages`**(*item*)

`ebooklib.utils.`**`get_pages_for_items`**(*items*)

`ebooklib.utils.`**`guess_type`**(*extenstion*)

`ebooklib.utils.`**`parse_html_string`**(*s*)

`ebooklib.utils.`**`parse_string`**(*s*)

4.4 Subpackages

4.4.1 plugins Package

`base` Module

class `ebooklib.plugins.base.`**`BasePlugin`**

Bases: `object`

`after_read`(*book*)
Processing after save

`after_write`(*book*)
Processing after save

`before_read`(*book*)
Processing before save

`before_write`(*book*)
Processing before save

`html_after_read`(*book*, *chapter*)
Processing HTML before read.

`html_before_write`(*book*, *chapter*)
Processing HTML before save.

`item_after_read`(*book*, *item*)
Process general item after read.

`item_before_write`(*book*, *item*)
Process general item before write.

`booktype` Module

class `ebooklib.plugins.booktype.`**`BooktypeFootnotes`**(*booktype_book*)

Bases: *`ebooklib.plugins.base.BasePlugin`*

`NAME = 'Booktype Footnotes'`

`html_before_write`(*book*, *chapter*)
Processing HTML before save.

class `ebooklib.plugins.booktype.`**`BooktypeLinks`**(*booktype_book*)

Bases: *`ebooklib.plugins.base.BasePlugin`*

`NAME = 'Booktype Links'`

`html_before_write`(*book*, *chapter*)
Processing HTML before save.

`sourcecode` Module

class `ebooklib.plugins.sourcecode.`**`SourceHighlighter`**

Bases: *`ebooklib.plugins.base.BasePlugin`*

`html_before_write`(*book*, *chapter*)
Processing HTML before save.

`standard` Module

class ebooklib.plugins.standard.**SyntaxPlugin**

Bases: *ebooklib.plugins.base.BasePlugin*

NAME = 'Check HTML syntax'

html_before_write(*book*, *chapter*)

Processing HTML before save.

ebooklib.plugins.standard.**leave_only**(*item*, *tag_list*)

`tidyhtml` Module

class ebooklib.plugins.tidyhtml.**TidyPlugin**(*extra={}*)

Bases: *ebooklib.plugins.base.BasePlugin*

NAME = 'Tidy HTML'

OPTIONS = {'char-encoding': 'utf8', 'tidy-mark': 'no'}

html_after_read(*book*, *chapter*)

Processing HTML before read.

html_before_write(*book*, *chapter*)

Processing HTML before save.

ebooklib.plugins.tidyhtml.**tidy_cleanup**(*content*, ***extra*)

CHAPTER 5

Indices and tables

- genindex
- modindex
- search

Python Module Index

Index