Python-Epub Documentation

Version 0.5.2

Florian Strzelecki

sept. 30, 2017

Table des matières

1 Le fichier Epub **1**
- 1.1 Introduction . . . 1
- 1.2 Ouvrir un fichier Epub . . . 1
- 1.3 Lire le contenu du fichier . . . 2
- 1.4 Écrire dans un fichier epub . . . 2
- 1.5 API du module . . . 3

2 Le fichier OPF **7**
- 2.1 Open Packaging Format . . . 7
- 2.2 Manipuler le fichier OPF . . . 8
- 2.3 API du module . . . 10

3 Le fichier de navigation NCX **15**
- 3.1 Navigation Center eXtended . . . 15
- 3.2 API du module . . . 17

4 Utilitaires **23**

5 Changelog **25**
- 5.1 Version 0.5.2 . . . 25
- 5.2 Version 0.5.1 . . . 25
- 5.3 Version 0.5.0 . . . 25
- 5.4 Version 0.4.0 . . . 26

6 Introduction **27**

7 Licence **29**

8 Installation **31**

9 Version et compatibilité **33**

10 Utilisation **35**

11 Indices and tables **37**

Index des modules Python **39**

CHAPITRE 1

Le fichier Epub

Introduction

Pour manipuler un fichier epub, il est nécessaire d'en comprendre la structure. Cela n'a rien de bien compliqué en réalité : il s'agit simplement d'une archive zip, avec une structure de contenu un peu particulière, et quelques règles supplémentaires sur le contenu (format, style, etc.).

Lorsque vous souhaitez lire un fichier epub, vous devez commencer par l'ouvrir comme un fichier zip, puis rechercher les fichiers importants qui vous permettront de naviguer au travers (le fichier *OPF* d'une part, et le fichier de navigation *NCX* d'autre part).

Le module *epub* vous permet de vous abstraire d'une grosse partie du travail d'analyse de cette structure, en fournissant des objets contenant ces informations.

Cependant, pour plus de facilité, il est vivement recommandé de connaître un minimum la spécification Epub 2, disponible en ligne sur le site de l'IDPF.

Ouvrir un fichier Epub

La fonction principale du module est *epub.open_epub()*, qui permet d'ouvrir un fichier epub et d'obtenir un objet de la classe *EpubFile*, permettant de manipuler les données du fichier : son contenu et ses meta-données.

Elle s'utilise très simplement en lui fournissant le chemin d'accès (relatif ou absolu) du fichier epub à ouvrir, et retourne un objet de la classe *epub.EpubFile* représentant l'archive epub ainsi ouverte.

On parle ici "d'archive epub", car un fichier epub n'est ni plus ni moins qu'un fichier zip avec une structure un peu particulière. Tous les détails de cette utilisation du format zip se trouvent dans la spécification epub (et plus spécifiquement dans la spécification OCF).

De plus, l'objet *EpubFile* implémentant les bonnes méthodes, vous pouvez utiliser la fonction *open_epub()* avec l'instruction `with` :

```
with epub.open_epub('path/to/my_book.epub') as book:
    print 'Vous pouvez lire votre livre !'
```

Lire le contenu du fichier

Suivant la norme Epub 2, le contenu du fichier epub est décrit par le fichier opf, indiqué par le fichier `META-INF/container.xml`. Si *open_epub()* se charge de le trouver pour vous, il vous reste à exploiter la liste des fichiers.

Pour se faire, vous pouvez, au choix, utiliser les informations du fichier opf (via l'attribut *EpubFile.opf*), ou les informations du fichier ncx (via l'attribut *EpubFile.toc*).

Par exemple pour accéder à l'ensemble des items du fichier :

```
book = epub.open_epub('book.epub')

for item in book.opf.manifest.values():
    # read the content
    data = book.read_item(item)
```

Il est possible d'accéder à l'ordre linéaire des éléments en utilisant l'objet de la classe *opf.Spine* disponible de cette façon :

```
book = epub.open_epub('book.epub')

for item_id, linear in book.opf.spine.itemrefs:
    item = book.get_item(item_id)
    # Check if linear or not
    if linear:
        print 'Linear item "%s"' % item.href
    else:
        print 'Non-linear item "%s"' % item.href
    # read the content
    data = book.read_item(item)
```

Quant au fichier de navigation NCX, il est accessible via l'attribut *EpubFile.toc*. Cet attribut est de la classe *ncx.Ncx* et représente le fichier de navigation du livre numérique, et propose une structure logique de lecture des fichiers (mais cela demande une connaissance plus approfondie de la structure d'un fichier epub).

Écrire dans un fichier epub

Nouveauté de la version 0.4.0, vous pouvez aussi ouvrir un fichier epub en mode écriture. Le mode *w* permet d'ouvrir un fichier epub vierge, et le mode *a* d'ajouter du contenu à un fichier epub existant.

Pour le moment, vous ne pouvez qu'ajouter du contenu à un fichier epub, et pas en retirer.

```
book = epub.open_epub('book.epub', u'w')
filename = 'path/to/file/to/add.xhtml'
manifest_item = epub.opf.ManifestItem(identifier='IdFile',
                                      href='path/into/epub/add.xhtml',
                                      media_type='application/xhtml+xml')
book.add_item(filename, manifest_item)
book.close()
```

Ce petit exemple vous montre le fonctionnement, qui reste très basique.

Le contenu du fichier epub est réellement sauvegardé lorsqu'il est fermé, c'est à dire à l'appel de la méthode *epub.EpubFile.close()*.

API du module

La fonction open_epub

epub.**open_epub**(*filename*, *mode='r'*)

Ouvre un fichier epub, et retourne un objet *epub.EpubFile*. Vous pouvez ouvrir le fichier en lecture seule (mode *r* par défaut) ou en écriture (mode *w* ou mode *a*).

Il est possible d'utiliser cette fonction avec la directive `with` de cette façon :

```
with epub.open_epub('path/to/my.epub') as book:
    # do thing with book, for exemple:
    print book.read_item('Text/cover.xhtml')
```

Le mode d'écriture *w* ouvre le fichier epub en écriture et considère un fichier vierge. Si le fichier existe déjà il est remplacé.

Le mode d'écriture *a* ouvre le fichier epub en écriture et permet de modifier un fichier déjà existant. Si le fichier n'existe pas, il est créé et traité de la même façon qu'avec le mode *w*.

Paramètres **filename** (*string*) – chemin d'accès au fichier epub

La classe EpubFile

class epub.**EpubFile**(*filename*)

Cette classe représente le contenu d'un fichier au format Epub. Elle permet de représenter tant les meta-données (le fichier OPF) que la navigation (le fichier NCX).

Les éléments du fichier epub ne sont pas tous directement accessibles : il faut passer par les attributs *opf* et *toc*.

Il est préférable d'utiliser la fonction *epub.open_epub()* plutôt qu'instancier directement un objet EpubFile.

opf

Objet de la classe *Opf* représentant le fichier opf.

opf_path

Chemin d'accès interne à l'archive zip au fichier OPF.

Le chemin du fichier OPF est spécifié par le fichier `META-INF/container.xml` et ce chemin est conservé dans cet attribut. L'emplacement sert de référence à l'emplacement des autres fichiers du livre (html, styles, images, etc.).

toc

Le sigle "toc" signifie "Table Of Content", et cet attribut représente le fichier ncx décrivant cette table des matières.

Il s'agit d'un objet de la classe *Ncx*, qui peut être accéder directement pour en utiliser le contenu.

uid

Identifiant unique du fichier epub. Cet identifiant peut être un ISBN ou un autre format d'identifiant unique.

Le format de cet attribut est un tuple, contenant trois éléments :
—la valeur de l'identifiant unique (`uid[0]`)
—l'identifiant associé dans le fichier OPF (`uid[1]`)
—le schéma tel que défini par l'attribut `opf::scheme` (`uid[2]`)
Cet identifiant peut être retrouvé dans la liste des identifiants via les meta-données (voir aussi *epub.opf.Metadata.identifiers*).

__init__ (*file*)
Initialise l'objet *epub.EpubFile*.

add_item (*filename*, *manifest_item*)
Permet d'ajouter un fichier au livre numérique.

Le premier paramètre indique le fichier source (le contenu), et le second indique les meta-données à ajouter au fichier OPF, dont l'emplacement dans l'archive epub.

L'attribut *id* du *manifest_item* ne doit pas déjà exister dans le fichier epub.

L'attribut *href* du *manifest_item* doit être un chemin d'accès relatif à l'emplacement du fichier OPF.

Paramètres
—**filename** (*string*) – Le chemin d'accès au fichier à ajouter.
—**manifest_item** (epub.opf.ManifestItem) – l'item décrivrant le fichier à ajouter pour le fichier OPF.

Lève
—**RuntimeError** – Si le fichier est déjà clos.
—**IOError** – Si le fichier n'est pas ouvert en écriture.

check_mode_write ()
Lève une exception si le fichier n'est pas ouvert en mode écriture (*w* ou *a*), ou si le fichier est déjà clos et qu'il ne peut être modifié.

Lève
—**RuntimeError** – Si le fichier est déjà clos.
—**IOError** – Si le fichier n'est pas ouvert en écriture.

close ()
Ferme le fichier epub. S'il était ouvert en mode écriture (*w* ou *a*), alors le fichier OPF et le fichier NCX sont générés en XML et modifié dans l'archive epub avant la fermeture.

L'appel à cette méthode assure la sauvegarde des modifications effectuées.

extract_item (*item*[, *to_path=None*])
Extrait le contenu d'un fichier présent dans l'archive epub à l'emplacement indiqué par *to_path*. Si *to_path* vaut *None* alors le fichier est extrait à l'emplacement de travail courrant.

Le paramètre `item` peut être un objet *epub.opf.ManifestItem* ou un chemin d'accès du fichier, chemin relatif à l'emplacement du fichier OPF.

Ce chemin ne doit pas contenir de fragment d'url (commençant pas #).

Voir aussi la méthode `zipfile.ZipFile.Extract()` sur laquelle repose le comportement de cette méthode.

http://docs.python.org/library/zipfile.html#zipfile.ZipFile.extract

Paramètres
—**item** (*mixed*) – Le chemin ou le Manifest Item.
—**to_path** (*string*) – Le chemin où extraire le fichier (par défaut il s'agit du répertoire de travail courrant).

get_item (*identifier*)
Cette fonction permet de récupérer un "item" du manifest. Le fichier opf décrit, via son "manifest", une liste des fichiers composant le livre numérique : chacun de ces éléments est un "item", qui représente l'un des fichier de l'epub.

Paramètres **identifier** (*string*) – Identifiant de l'item recherché.

Type retourné *epub.opf.ManifestItem* ou `None` s'il n'existe pas.

get_item_by_href (*href*)
Fonctionne de la même façon que *get_item* en utilisant la valeur de l'attribut *href* des items du manifest.

Paramètres **href** (*string*) – Chemin d'accès (relatif au fichier opf) de l'item recherché.

Type retourné *epub.opf.ManifestItem* ou `None` s'il n'existe pas.

read_item(*item*)

Retourne le contenu d'un fichier présent dans l'archive epub.

Le paramètre `item` peut être un objet *epub.opf.ManifestItem* ou un chemin d'accès du fichier, chemin relatif à l'emplacement du fichier OPF.

Ce chemin ne doit pas contenir de fragment d'url (commençant pas #).

Voir aussi la méthode `zipfile.ZipFile.read()`.

http://docs.python.org/library/zipfile.html#zipfile.ZipFile.read

```
book = epub.open_epub('mybook.epub')

# get chapter 1
item = book.get_item('chap01')
item_path = item.href # 'Text/chap01.xhtml'

# display the same thing
print book.read_item(item)
print book.read_item(item_path)

# but this won't work!
content = book.read_item('Text/chap01.xhtml#part1')
```

Paramètres **item** (*mixed*) – Le chemin ou le Manifest Item.

Type retourné string

La classe Book

class `epub.`**Book**

Cette classe permet de simplifier l'accès en lecture à un fichier epub. Un objet Book sert de proxy à l'objet plus complexe EpubFile, par un ensemble de *@property* adaptées.

Note : Cette classe n'est pas encore prête à être employée en production. À utiliser avec précaution.

CHAPITRE 2

Le fichier OPF

Open Packaging Format

Le format OPF (pour *Open Packaging Format*) spécifié par l'IDPF permet d'indiquer au système de lecture quelle est la structure et le contenu d'un fichier epub.

Ses principaux composants sont ses meta-données et son élément `<manifest>`, ce dernier référençant les fichiers qui composent effectivement le livre numérique.

Différents éléments annexes sont aussi présents : l'élément `<spine>` qui donne un ordre de lecture linéaire, et l'élément `<guide>` qui référence les différentes tables des matières, des illustrations, etc.

La bibliothèque python-epub propose un module à part entière pour manipuler ce format (dans sa version pour Epub 2.0), permettant une plus grande souplesse dans son utilisation.

Chaque élément du fichier OPF est représenté par une structure permettant d'accéder à tous ses éléments, sans avoir à analyser le fichier xml soi-même. Ces éléments sont tous renseignés dans les attributs de la classe *`Opf`* :

- *`Opf.manifest`* pour l'élément `<manifest>`
- *`Opf.metadata`* pour l'élément `<metadata>`
- *`Opf.guide`* pour l'élément `<guide>` (s'il est présent)
- *`Opf.spine`* pour l'élément `<spine>`

L'élément `<manifest>`

Cet élément référence la liste des fichiers du livre numérique : textes, images, feuilles de style, couverture, etc. ainsi que les *fallback* des fichiers qui sortent de la spécification Epub (comme les fichiers PDF).

Vous pouvez obtenir plus d'information directement dans la spécification epub à propos de l'élément manifest.

Il est représenté par la classe *`Manifest`*, et chaque élément du manifest est représenté par un objet de la classe *`ManifestItem`*. En outre, la classe *`Manifest`* peut être utilisée exactement comme un `dict` ne pouvant contenir des objets de type `ManifestItem`.

Les métadonnées et l'élément <metadata>

Les méta-données d'un epub sont renseignés dans l'élément <metadata> du fichier OPF. Pour les représenter, un objet de la classe *Metadata* est employé.

La description de chacune de ces meta-données est disponible dans la spécification Epub, section "Metadata".

Comme la pluspart des meta-données peuvent être renseignées plusieurs fois, les attributs de cette classe sont souvent des listes d'éléments (principalement des tuples contenant à leur tour de simples chaînes de caractères).

Par exemple, pour l'élément <title> qui peut se décliner en plusieurs langues, voici comment il est possible de l'exploiter :

```
# meta est un objet de la classe Metadata contenant plusieurs titres
for title, lang in meta.titles:
    print 'Le titre en %s est "%s"' % (title, lang)
```

Chaque attribut est décrit avec la forme de son contenu dans la documentation de la classe *Metadata*.

L'élément <guide>

L'élément <guide> d'un fichier OPF représente une liste des tables et des références du livre, pouvant indiquer la couverture, la table des contenus, des illustrations, etc.

Voir aussi la spécification epub OPF, section "guide"

Cet élément est représenté par la classe *Guide*.

L'élément <spine>

L'élément <spine> propose une liste de fichiers dans un ordre de lecture dit "linéaire", c'est à dire dans l'ordre de lecture logique.

La spécification epub OPF, section "spine" donne plus d'information au sujet de cet élément.

C'est aussi à partir de cet élément que l'on obtient l'identifiant du fichier de navigation NCX, qui permet de retrouver le fichier dans la liste du manifest.

Cet élément est représenté par la classe *Spine*.

Manipuler le fichier OPF

En connaissant la structure d'un fichier OPF, structure décrite dans la spécification Epub pour le format OPF, il est plutôt simple d'exploiter les données proposées par la classe *Opf*.

Cependant, lire une spécification entière n'est pas forcément nécessaire... passons à des explications concrètes : comment manipuler un fichier OPF avec le module *epub.opf* ?

Ouvrir et analyser un fichier OPF

Le plus simple est d'utiliser la fonction *parse_opf()*, en lui fournissant le contenu du fichier, sous forme d'une chaîne de caractère. Cette fonction retourne alors un objet *Opf* qu'il suffit d'utiliser.

Cet objet permet d'accéder aux différents éléments via ses attributs : *metadata*, *manifest*, *spine*, et *guide*.

Obtenir la liste des fichiers

C'est l'élément `<manifest>` qui propose ces informations, il est représenté par un objet de la classe *Manifest*, classe qui étend le comportement du type `dict` :

```
# manifest est un objet de la classe epub.opf.Manifest
for id in manifest:
    # item est un objet de la classe ManifestItem
    item = manifest[id]
    print 'Fichier Id : "%s" [href="%s"]' % (item.id, item.href)
```

À partir d'un objet de la classe *ManifestItem*, un objet de la classe *epub.EpubFile* peut retrouver le contenu associé, grâce à sa méthode `epub.EpubFile.read()`.

Les meta-données

La classe *Metadata* permet de représenter et donc de manipuler les meta-données d'un fichier epub : chacun de ses attributs représente un type de meta-données.

Les règles suivantes s'appliquent à tous les attributs composés de plusieurs éléments :

- La valeur d'une meta-donnée est représentée par un tuple de ses attributs, chacun représenté par une chaîne de caractère
- Une meta-donnée peut être présente plusieurs fois avec des valeurs différentes : chacune est alors stockée dans une liste
- Un attribut qui n'est pas renseigné dans le fichier xml est représenté par une chaîne vide.

Ainsi, l'attribut `titles` est une `list` de `tuple` de la forme `(title, lang)`.

Les autres attributs simples sont représentées par une chaîne de caractères.

```
"""
<metadata>
    <dc:title xml:lang="fr">Titre français</dc:title>
    <dc:title xml:lang="en">English title</dc:title>
</metadata>
"""

# equivalent metadata
metadata = epub.opf.Metadata()
metadata.title = [('Titre français', 'fr'), ('English title', 'en')]
```

Utiliser l'élélement `<spine>`

L'élément `<spine>` ne fournit pas directement une liste de fichiers, mais y fait seulement référence par l'identifiant de ces fichiers.

```
# spine est un objet de la classe epub.opf.Spine
for id, linear in spine.itemrefs:
    # item est un objet de la classe ManifestItem
    item = book.get_item(id)
    print 'Fichier Id : "%s" [href="%s"]' % (item.id, item.href)
```

API du module

La fonction `parse_opf`

epub.opf.**parse_opf**(*xml_string*)

Analyse les données xml au format OPF, et retourne un objet de la classe *Opf* représentant ces données.

Paramètres **xml_string** (*string*) – Le contenu du fichier xml OPF.

Type retourné *Opf*

La classe `Opf`

class epub.opf.**Opf**(*uid_id=None*, *version='2.0'*, *xmlns=XMLNS_OPF*, *metadata=None*, *manifest=None*, *spine=None*, *guide=None*)

Paramètres

- **metadata** (epub.opf.Metadata) – Les méta-données du fichier OPF
- **manifest** (epub.opf.Manifest) – L'élélement manifest du fichier OPF
- **spine** (epub.opf.Spine) – L'élément spine du fichier OPF
- **guide** (epub.opf.Guide) – L'élément guide du fichier OPF

uid_id

Identifiant de l'identifiant unique du livre numérique. L'identifiant ainsi référencé est disponible dans la liste des identifiants des méta-données (voir l'attribut *metadata* et son attribut *Metadata.identifiers*).

version

Indique la version du fichier opf (2.0 par défaut), sous la forme d'une chaîne de caractère.

xmlns

Indique le namespace du fichier OPF. Cette valeur ne devrait pas être modifiée.

Sa valeur par défaut est `http://www.idpf.org/2007/opf`.

metadata

Représente les méta-données du fichier epub, sous la forme d'un objet de la classe *Metadata*.

manifest

Objet de la classe *Manifest* représentant la balise `<manifest>` du fichier OPF référençant les fichiers du livre numérique.

spine

Objet de la classe *Spine* représentant la balise `<spine>` du fichier OPF indiquant un ordre de lecture linéaire.

guide

Objet de la classe *Guide* représentant la balise `<guide>` du fichier OPF indiquant une liste de références (tables de contenus, d'illustration, etc.).

La classe Metadata

class epub.opf.**Metadata**

titles

Liste des éléments `<dc:title>` des meta-données. Chaque élément de la liste est un tuple de la forme `(title, lang)`.

creators

Liste des éléments `<dc:creator>` des meta-données. Chaque élément de la liste est un tuple de la forme `(name, role, file as)`.

subjects
Liste des éléments <dc:subjet> des meta-données. Chaque élément de la liste est une chaîne de caractère représentant la valeur du sujet.

description
L'élément <dc:description>, représenté par une chaîne de caractère.

publisher
L'élément <dc:publisher>, représenté par une chaîne de caractère.

contributors
Liste des éléments <dc:contributor> des meta-données. Chaque élément de la liste est un tuple de la forme `(name, role, file as)`.

dates
Liste des éléments <dc:date> des meta-données. Chaque élément de la liste est un tuple de la forme `(date, event)`.

dc_type
L'élément <dc:type>, représenté par une chaîne de caractère.

format
L'élément <dc:format>, représenté par une chaîne de caractère.

identifiers
Liste des éléments <dc:identifier> des meta-données. Chaque élément de la liste est un tuple de la forme `(uid, identifier, scheme)`.

La partie `identifier` est l'identifiant qui permet de référencer quel `identifier` doit être utilisé pour définir l'UID de fichier epub (c'est l'identifiant référencé par l'attribut `unique-identifier` du fichier opf).

source
L'élément <dc:source>, représenté par une chaîne de caractère.

languages
Liste des éléments <dc:language> des meta-données. Chaque élément de la liste est une chaîne de caractère.

Plus de précision sur la balise <dc:language> dans la spécification epub, section "metadata : language"

relation
L'élément <dc:relation>, représenté par une chaîne de caractère.

coverage
L'élément <dc:coverage>, représenté par une chaîne de caractère.

right
L'élément <dc:rights>, représenté par une chaîne de caractère.

metas
Liste des éléments <dc:meta> des meta-données. Chaque élément de la liste est un tuple de la forme `(name, content)`.

add_title(*title*, *lang=''*)

Paramètres
—**title** (*string*) –
—**lang** (*string*) –

add_creator(*name*, *role='aut'*, *file_as=''*)

Paramètres
—**name** (*string*) –
—**role** (*string*) –
—**file_as** (*string*) –

add_subject(*subject*)

Paramètres**subject** (*string*) –

add_contributor(*name*, *role='oth'*, *file_as=''*)

Paramètres
—**name** (*string*) –
—**role** (*string*) –
—**file_as** (*string*) –

add_date(*date*, *event=''*)

Paramètres
—**date** (*string*) –
—**event** (*string*) –

add_identifier(*content*, *identifier=''*, *scheme=''*)

Paramètres
—**content** (*string*) –
—**identifier** (*string*) –
—**scheme** (*string*) –

add_language(*lang*)

Paramètres**lang** (*string*) –

add_meta(*name*, *content*)

Paramètres
—**name** (*string*) –
—**content** (*string*) –

as_xml_element()
Retourne un élément xml <manifest> équivalent au contenu de l'objet.

Type retournéxml.dom.Element

get_isbn()
Retourne l'identifiant de type isbn, qui doit être renseigné dans la liste *identifiers*.

Les classes Manifest et ManifestItem

class epub.opf.**Manifest**
La classe *Manifest* étend la classe collection.OrderedDict et peut donc être utilisé de la même façon. Cependant, lorsqu'un élément est inséré dans le dictionnaire, il est vérifié que l'élément en question dispose d'au moins deux attributs nécessaires : identifier et href.

Il est préférable de ne stocker que des objets de la classe *ManifestItem*, qui correspond à un usage "normal".

La clé d'accès à chaque élément est la valeur de son attribut identifier, ce qui permet de retrouver rapidement un objet dans le manifest, exemple :

```
# manifest is an epub.opf.Manifest object
item = manifest['chap001']
print item.identifier # display "chap001"

item in manifest # Return true

# raise a Value Error (key != item.identifier)
manifest['bad_id'] = item
```

add_item(*identifier*, *href*, *media_type=None*, *fallback=None*, *required_namespace=None*, *required_modules=None*, *fallback_style=None*)
Crée et ajoute un élément au manifest.

Cette méthode n'ajoute rien d'autre qu'une référence à un fichier, mais en aucun cas ne permet d'ajouter concrètement un fichier à l'archive epub : la classe OPF permet de gérer uniquement le fichier XML, et ne se préoccupe donc pas du contenu réel du fichier epub.

Paramètres

—**identifier** (*string*) – Identifiant
—**href** (*string*) – Chemin d'accès du fichier, relatif à l'emplacement du fichier OPF
—**media_type** (*string*) – Le mime-type de l'élément.
—**fallback** (*string*) – Identifiant de l'élément fallback
—**required_namespace** (*string*) – voir spec epub "required-namespace"
—**required_module** (*string*) – voir spec epub "required-module"
—**fallback_style** (*string*) – Identifiant de l'élément de style en fallback.

append(*item*)
Ajoute un élément au manifest. Cet élément doit avoir au moins deux attributs : `identifier` et `href`. L'attribut `identifier` doit être une chaîne unicode.

Paramètresitem (`epub.opf.ManifestItem`) – l'élément à ajouter au manifest

as_xml_element()
Retourne un élément xml `<manifest>` équivalent au contenu de l'objet.

Type retourné`xml.dom.Element`

class `epub.opf.`**ManifestItem**(*identifier*, *href*, *media_type=None*, *fallback=None*, *required_namespace=None*, *required_modules=None*, *fallback_style=None*)
Un objet de la classe *ManifestItem* représente un élément du manifest du fichier epub, c'est à dire l'un des fichiers qui compose le livre numérique.
Chacun de ses attributs représente l'un des attributs de l'élément `<item>` tel qu'il est décrit par la spécification epub.

```
"""
<item id="chap01" href="Text/chap01.xhtml" media-type="application/xhtml+xml"/>
"""

# equivalent metadata
item = epub.opf.ManifestItem()
item.identifier = 'chap01'
item.href = 'Text/chap01.xhtml'
item.media_type = 'application/xhtml+xml'

# ou bien directement avec le constructeur
item = epub.opf.ManifestItem(identifier='chap01', href='Text/chap01.xhtml',
                             media_type='application/xhtml+xml')
```

identifier
Identifiant de l'item, qui doit être unique pour permettre de récupérer chaque élément dans la liste des items du manifest.
Il s'agit d'une chaîne unicode.

href
Chemin d'accès au fichier présent dans l'archive zip. Ce chemin d'accès est relatif à l'emplacement du fichier opf dans lequel est décrit l'item.

media_types
Chaîne de caractère, indique le mime-type du fichier correspondant à l'item.

fallback
Chaîne de caractère, indique l'identifiant de l'item servant de *fallback* à cet item (ce mécanisme est décrit dans la spécification epub).

required_namespace
Chaîne de caractère, indique le namespace pour les élements *"Out-of-Line XML Island"*.

required_modules
Chaîne de caractère, indique le ou les modules pour les élements *"Out-of-Line XML Island"*.

fallback_style
Indique l'identifiant de l'item servant de *fallback* pour la feuille de style à cet item (ce mécanisme est décrit dans la spécification epub).

as_xml_element()
Créer un `Element Node` représentant l'objet avec ses attributs. Un attribut dont la valeur est `None` ou une chaîne vide ne sera pas ajouté à l'élément xml retourné (il ne crée pas d'attribut vide).

Type retourné `xml.dom.Element`

Les classes Guide et Spine

class `epub.opf.`**Guide**

references
Liste des références de l'élément `<guide>`. Chaque élément de la liste est un `tuple` de la forme `(href, ref_type, title)`.
La valeur de `href` est une url d'accès relative à l'emplacement du fichier OPF. Cependant, cette url ne peut pas être utilisée directement par les méthodes *epub.EpubFile.get_item_by_href()* ou `epub.EpubFile.read()`, car il peut comporter une ancre (le caractère `#` suivit d'un identifiant d'ancre).

add_reference(*href*, *ref_type=None*, *title=None*)
Ajoute une référence au guide.

Paramètres
—**href** (*string*) – Chemin d'accès du fichier, relatif à l'emplacement du fichier OPF, peut contenir une ancre
—**type** (*string*) – Le type de fichier (voir la spécification epub)
—**title** (*string*) – Titre de la référence

append(*reference*)
Ajoute une référence au guide ; elle doit être un tuple de la forme `(href, ref_type, title)`.

Paramètres **reference** (*tuple*) – la référence à ajouter.

class `epub.opf.`**Spine**

itemrefs
Liste des items du manifest référencés par l'ordre de lecture de l'élément `<spine>`. Les items sont dans l'ordre naturel de la liste (c'est à dire que l'élement présent en première position est le premier dans l'ordre de lecture).
Chaque élement est un `tuple` de la forme `(idref, linear)`. La valeur de `idref` est une chaîne de caractère indiquant l'identifiant du fichier listé dans le manifest, et `linear` est un booléen indiquant si l'élément sort du flux de lecture "linéaire" (voir la spécification epub pour plus d'information à ce sujet).

toc
Chaîne de caractère : il s'agit de l'identifiant permettant de récupérer le fichier de navigation NCX dans la liste des fichiers du manifest.

append(itemref):
Ajoute une référence au spine ; elle doit être un tuple de la forme `(identifier, linear)`. L'élément est ajouté à la suite des autres.

Paramètres **itemref** (*tuple*) – la référence à ajouter.

add_itemref(*idref*, *linear=True*)
Ajoute un élément au spine, à la suite des autres déjà présents.

Paramètres
—**idref** (*string*) – l'identifiant de l'élément à ajouter
—**linear** (*bool*) – indicateur d'élément linéaire (par défaut) ou non

CHAPITRE 3

Le fichier de navigation NCX

Navigation Center eXtended

Le format NCX est un format XML qui permet de décrire la structure de navigation d'un livre numérique. La spécification de ce format est dirigé et maintenu par le DAISY Consortium.

> The Navigation Control file for XML applications (NCX) exposes the hierarchical structure of a Publication to allow the user to navigate through it. The NCX is similar to a table of contents in that it enables the reader to jump directly to any of the major structural elements of the document

Il est employé dans le cadre du format epub avec quelques modifications apportées par la spécification epub OPF.

Tout comme le format OPF, la bibliothèque python-epub propose un module à part entière pour manipuler ce format plus simplement.

La classe *Ncx* permet au développeur d'utiliser ce contenu au travers de ses différents attributs :

- *Ncx.nav_map* pour l'élément <navMap>
- *Ncx.page_list* pour l'élément <pageList>
- *Ncx.nav_lists* pour les éléments <navList>

Comprendre la structure

La clé pour comprendre comment utiliser le format NCX est sans doute de comprendre le principe d'une table de matière ; prenez l'exemple suivant :

- **Volume 1**
  - **Partie I**
    - Chapitre 1
    - Chapitre 2
    - Chapitre 3
  - **Partie II**
    - Chapitre 1
    - Chapitre 2
    - Chapitre 3

Cette structure est finalement très simple : un arbre aux ramification descendantes. Cette liste est une `<navMap>`, dont chaque élément est un `<navPoint>`.

Voici la même chose au format xml :

```
<navMap>
    <navPoint playOrder="1" id="vol1">
        <navLabel><text>Volume 1</text></navLabel>
        <content src="Text/vol1.html"/>
        <navPoint playOrder="2">
            <navLabel><text>Partie I</text></navLabel>
            <content src="Text/vol1/part1.html"/>
            <navPoint playOrder="3">
                <navLabel><text>Chapitre 1</text></navLabel>
                <content src="Text/vol1/part1.html#chap1"/>
            </navPoint>
            <navPoint playOrder="4">
                <navLabel><text>Chapitre 2</text></navLabel>
                <content src="Text/vol1/part1.html#chap2"/>
            </navPoint>
            <navPoint playOrder="5">
                <navLabel><text>Chapitre 3</text></navLabel>
                <content src="Text/vol1/part1.html#chap3"/>
            </navPoint>
        </navPoint>
        <navPoint playOrder="6">
            <navLabel><text>Partie II</text></navLabel>
            <content src="Text/vol1/part2.html"/>
            <navPoint>
                <navLabel><text>Chapitre 1</text></navLabel>
                <content src="Text/vol1/part2/chap1.html"/>
            </navPoint>
            <navPoint>
                <navLabel><text>Chapitre 2</text></navLabel>
                <content src="Text/vol1/part2/chap2.html"/>
            </navPoint>
            <navPoint>
                <navLabel><text>Chapitre 3</text></navLabel>
                <content src="Text/vol1/part2/chap3.html"/>
            </navPoint>
        </navPoint>
    </navPoint>
</navMap>
```

Vous aurez sans doute remarqué plusieurs choses :

- L'attribut `playOrder` est global : il indique l'ordre de lecture
- Cet attribut est cependant **optionnel**
- Un `<navPoint>` représente une entrée de la navigation
- Un `<navPoint>` peut avoir lui-même plusieurs `<navPoint>` en fils
- Le contenu pointé par un `<navPoint>`, via sa balise `<content>` n'est pas un chemin d'accès mais une url relative à l'emplacement du fichier NCX

Armée de cette nouvelle compréhension, utiliser un objet *`Ncx`* ne devrait plus être un gros problème.

API du module

La fonction `parse_ncx`

epub.ncx.**parse_ncx**(*xml_string*)

Analyse les données xml au format NCX, et retourne un objet de la classe *Ncx* représentant ces données.

Paramètres **xml_string** (*string*) – Le contenu du fichier xml NCX.

Type retourné *Ncx*

La classe `Ncx`

class epub.ncx.**Ncx**

Représente le fichier NCX d'un livre numérique. Un fichier NCX est un fichier xml respectant les spécifications de la norme NCX avec les modifications apportées par la spécification Epub.

xmlns
Namespace utilisé pour le document NCX, dont la valeur devrait toujours être `'http://www.daisy.org/z3986/2005/ncx/'`.

version
Version du fichier NCX, dont la valeur devrait toujours être `'2005-1'`.

lang
Langue du contenu du fichier NCX.

uid
Identifiant unique du livre.

depth
Représente la meta-donnée `dtb:depth`.

total_page_count
Représente la meta-donnée `dtb:totalPageCount`.

max_page_number
Représente la meta-donnée `dtb:maxPageNumber`.

generator
Représente la meta-donnée `dtb:generator`.

title
Titre du livre.

authors
Liste des auteurs du livre.

nav_map
Objet de la classe *NavMap* représentant l'élément `<navMap>` du fichier NCX. Cet attribut permet d'accéder à la structure de navigation principale.

page_list
Objet de la classe *PageList* représentant l'élément `<pageList>` du fichier NCX.

nav_lists
Liste d'objets de la classe *NavList* représentant les éléments `<navList>` du fichier NCX.

Il peut n'y avoir aucun élément dans cette liste.

add_nav_list(*nav_list*)
Ajoute un object *NavList* à la liste des `NavList`.

Paramètres **nav_list** (`epub.ncx.NavList`) – la liste à ajouter

as_xml_document()
Retourne l'élément XML Dom correspondant à la structure de l'objet.

Type retourné `xml.dom.Element`

Les classes `NavMap` et `NavPoint`

class epub.ncx.**NavMap**

identifier
Identifiant de la NavMap. Chaîne de caractère (peut être vide).

labels
Liste des labels de la NavMap : chaque label et un tuple de la forme `(label, lang, dir)`, indiquant respectivement le titre du label, sa langue, et la direction d'écriture (`ltr` ou `rtl`).

infos
Liste des infos de la NavMap : chaque info et un tuple de la forme `(info, lang, dir)`, indiquant respectivement le contenu de l'info, sa langue, et la direction d'écriture (`ltr` ou `rtl`).

Une "info" est simplement une description de l'élément.

nav_point
Liste des éléments `<navPoint>` en fils direct de l'élément `<navMap>` (et pas ses petits fils). Chaque élément de cette liste est un objet de la classe *NavPoint*.

add_label(*label*, *lang=''*, *direction=''*)

Paramètres
—**label** (*string*) – Texte de l'élément `navLabel`.
—**lang** (*string*) – Langue de l'élément.
—**direction** (*string*) – Direction du texte `rtl` ou `ltr`.

add_info(*label*, *lang=''*, *direction=''*)

Paramètres
—**label** (*string*) – Texte de l'élément `navInfo`.
—**lang** (*string*) – Langue de l'élément.
—**direction** (*string*) – Direction du texte `rtl` ou `ltr`.

add_point(*point*)
Ajoute un objet *NavPoint* à la liste des `navPoint`.

Paramètres **point** (`epub.ncx.NavPoint`) – ajoute un `NavPoint` à la liste de `navPoint`.

as_xml_document()
Retourne l'élément XML Dom correspondant à la structure de l'objet.

Type retourné `xml.dom.Element`

class epub.ncx.**NavPoint**

identifier
Chaîne de caractère, identifiant du `<navPoint>`.

class_name
Chaîne de caractère, indique la classe css proposée.

play_order
Chaîne de caractère, indique le placement dans l'ordre de lecture de l'élément. Peut être vide.

labels
Liste des labels du `<navPoint>` : chaque label et un tuple de la forme `(label, lang, dir)`, indiquant respectivement le titre du label, sa langue, et la direction d'écriture (`ltr` ou `rtl`).

src
Chaîne de caractère, indique l'url relative à l'emplacement du fichier NCX, et pouvant pointer vers des fragments de fichiers du fichier epub.

Exemple : `Text/chap1.xhtml#p36` indique le fichier `Text/chap1.xhtml` et plus spéficiquement à l'emplacement du fragment `p36`.

nav_point
Liste des éléments `<navPoint>` fils directs. Chaque élément est un objet de la classe *NavPoint*.

add_label(*label*, *lang=''*, *direction=''*)

Paramètres
—**label** (*string*) – Texte de l'élément navLabel.
—**lang** (*string*) – Langue de l'élément.
—**direction** (*string*) – Direction du texte rtl ou ltr.

add_point(*point*)
Ajoute un objet *NavPoint* à la liste des navPoint.

Paramètres**point** (epub.ncx.NavPoint) – ajoute un NavPoint à la liste de navPoint.

as_xml_document()
Retourne l'élément XML Dom correspondant à la structure de l'objet.

Type retournéxml.dom.Element

Les classes PageList et PageTarget

class epub.ncx.**PageList**

identifier
Chaîne de caractère, identifiant du <pageList>.

class_name
Chaîne de caractère, indique la classe css proposée.

labels
Liste des labels du <navPoint> : chaque label et un tuple de la forme (label, lang, dir), indiquant respectivement le titre du label, sa langue, et la direction d'écriture (ltr ou rtl).

infos
Liste des infos de la NavMap : chaque info et un tuple de la forme (info, lang, dir), indiquant respectivement le contenu de l'info, sa langue, et la direction d'écriture (ltr ou rtl).
Une "info" est simplement une description de l'élément.

page_target
Liste des éléments <pageTarget> fils directs. Chaque élément est un objet de la classe *PageTarget*.

add_label(*label*, *lang=''*, *direction=''*)

Paramètres
—**label** (*string*) – Texte de l'élément navLabel.
—**lang** (*string*) – Langue de l'élément.
—**direction** (*string*) – Direction du texte rtl ou ltr.

add_info(*label*, *lang=''*, *direction=''*)

Paramètres
—**label** (*string*) – Texte de l'élément navInfo.
—**lang** (*string*) – Langue de l'élément.
—**direction** (*string*) – Direction du texte rtl ou ltr.

add_target(*page_target*)
Ajoute un élément *PageTarget* à la liste des pageTarget.

Paramètres**page_target** (epub.ncx.PageTarget) – l'élément à ajouter

as_xml_document()
Retourne l'élément XML Dom correspondant à la structure de l'objet.

Type retournéxml.dom.Element

class epub.ncx.**PageTarget**

identifier
Chaîne de caractère, identifiant du <pageList>.

labels
Liste des labels du <navPoint> : chaque label et un tuple de la forme (label, lang, dir), indiquant respectivement le titre du label, sa langue, et la direction d'écriture (ltr ou rtl).

value
Chaîne de caractères, représente l'attribut value de l'élément.

target_type
Chaîne de caractères.

class_name
Chaîne de caractère, indique la classe css proposée.

play_order
Chaîne de caractère, indique le placement dans l'ordre de lecture de l'élément. Peut être vide.

src
Chaîne de caractère, indique l'url relative à l'emplacement du fichier NCX, et pouvant pointer vers des fragments de fichiers du fichier epub.

Exemple : Text/chap1.xhtml#p36 indique le fichier Text/chap1.xhtml et plus spéficiquement à l'emplacement du fragment p36.

add_label(*label*, *lang=''*, *direction=''*)

Paramètres
—**label** (*string*) – Texte de l'élément navLabel.
—**lang** (*string*) – Langue de l'élément.
—**direction** (*string*) – Direction du texte rtl ou ltr.

as_xml_document()
Retourne l'élément XML Dom correspondant à la structure de l'objet.

Type retournéxml.dom.Element

Les classes NavList et NavTarget

class epub.ncx.**NavList**

identifier
Chaîne de caractère, identifiant du <navList>.

class_name
Chaîne de caractère, indique la classe css proposée.

labels
Liste des labels du <navPoint> : chaque label et un tuple de la forme (label, lang, dir), indiquant respectivement le titre du label, sa langue, et la direction d'écriture (ltr ou rtl).

infos
Liste des infos de la NavMap : chaque info et un tuple de la forme (info, lang, dir), indiquant respectivement le contenu de l'info, sa langue, et la direction d'écriture (ltr ou rtl).

Une "info" est simplement une description de l'élément.

nav_target
Liste des éléments <navTarget> fils directs. Chaque élément est un objet de la classe *NavTarget*.

add_label(*label*, *lang=''*, *direction=''*)

Paramètres
—**label** (*string*) – Texte de l'élément navLabel.
—**lang** (*string*) – Langue de l'élément.
—**direction** (*string*) – Direction du texte rtl ou ltr.

add_info(*label*, *lang=''*, *direction=''*)

Paramètres
—**label** (*string*) – Texte de l'élément navInfo.

—**lang** (*string*) – Langue de l'élément.
—**direction** (*string*) – Direction du texte `rtl` ou `ltr`.

add_target(*nav_target*)
Ajoute un élément *NavTarget* à la liste des `navTarget`.

Paramètres **nav_target** (`epub.ncx.NavTarget`) – l'élément à ajouter

as_xml_document()
Retourne l'élément XML Dom correspondant à la structure de l'objet.

Type retourné `xml.dom.Element`

class `epub.ncx.`**NavTarget**

identifier
Chaîne de caractère, identifiant du `<pageList>`.

labels
Liste des labels du `<navPoint>` : chaque label et un tuple de la forme `(label, lang, dir)`, indiquant respectivement le titre du label, sa langue, et la direction d'écriture (`ltr` ou `rtl`).

value
Chaîne de caractères, représente l'attribut `value` de l'élément.

class_name
Chaîne de caractère, indique la classe css proposée.

play_order
Chaîne de caractère, indique le placement dans l'ordre de lecture de l'élément. Peut être vide.

src
Chaîne de caractère, indique l'url relative à l'emplacement du fichier NCX, et pouvant pointer vers des fragments de fichiers du fichier epub.

Exemple : `Text/chap1.xhtml#p36` indique le fichier `Text/chap1.xhtml` et plus spéficiquement à l'emplacement du fragment `p36`.

add_label(*label*, *lang=''*, *direction=''*)

Paramètres
—**label** (*string*) – Texte de l'élément `navLabel`.
—**lang** (*string*) – Langue de l'élément.
—**direction** (*string*) – Direction du texte `rtl` ou `ltr`.

as_xml_document()
Retourne l'élément XML Dom correspondant à la structure de l'objet.

Type retourné `xml.dom.Element`

CHAPITRE 4

Utilitaires

Pour des raisons pratiques, le module *epub* propose un module utilitaire appelé *epub.utils*. Il regroupe les fonctions pratiques à utilisées.

epub.utils.**get_node_text**(*node*)
Retourne le contenu texte d'un noeud XML de type *ELEMENT_NODE*. Si le texte est vide (le tag est vide), la valeur de retour sera une chaîne vide.

Paramètres **node** – Le noeud XML dont on cherche à récupérer le texte.

Ptype node xml.dom.Element

Type retourné string

epub.utils.**get_urlpath_part**(*url*)
Découpe une url en deux parties : l'url sans fragment, et le fragment. S'il n'y a pas de fragment alors l'url est retournée telle qu'elle avec fragment à *None*.

```
url = 'text/chapter1.xhtml#part2'
href, fragment = get_urlpath_part(url)
print href # 'text/chapter1.xhtml'
print fragment # '#part2'
```

Paramètres **url** (*string*) – Le chemin d'un fichier à décomposer en deux parties.

Type retourné tuple

CHAPITRE 5

Changelog

Version 0.5.2

Note : La fonctionnalité d'écriture d'un fichier Epub reste complexe à maîtriser et doit être considérée comme étant encore un mode "beta".

— Correction d'un bug où certains fichiers de base étaient dupliqués (par exemple le fichier mimetype).
— Correction d'un bug sous Windows concernant les chemins d'accès. Il est possible que d'autres usages d'os.path rendent la bibliothèque incompatible avec Windows (c'est un point qui sera amélioré dans une prochaine version).
— Correction d'un bug permettant de lire des epubs utilisant (probablement à tort) le namespace "opf" dans sa description des méta-données.

Version 0.5.1

— Il est désormais possible d'ouvrir un fichier epub qui ne contient pas de fichier NCX. Un warning est levé lorsqu'un tel fichier epub est ouvert.
— Il est désormais possible d'ouvrir un fichier epub n'ayant pas d'UID. Un warning est levé lorsqu'un tel fichier epub est ouvert.

Version 0.5.0

De nouvelles amélioration, et quelques modifications :
— Ajout de la méthode `epub.EpubFile.check_mode_write()` qui lève une exception si le fichier n'est pas ouvert en écriture.
— Support de python 2.6 avec l'emploi du module *ordereddict* disponible sur pypi (http://pypi.python.org/pypi/ordereddict) (il manque des tests pour être certain du bon fonctionnement avec Python 2.6).

— Nouvelle classe *epub.Book* servant de proxy pour simplifier et abstraire la manipulation du format epub. *Fonctionnalité expérimentale*.
— Ajout de la méthode `epub.EpubFile.extract_item()` qui reprend le même principe que `read_item()` en l'appliquant à l'exctraction de fichiers.
— Ajout de la fonction `epub.utils.get_urlpath_part()` permettant d'obtenir les deux parties des chemins des fichiers utilisés (entre autre) dans le fichier NCX ou l'élément spine du fichier OPF.
— Support de Python 3.2, avec des test-unitaires passant avec Python 2.7 et Python 3.2.
— Retrait de la notation des chaînes unicodes (le *u* devant les chaînes de caractères est retiré) dans la documentation.
— Ajout d'un warning sur l'usage de la fonction *epub.open()* : fonction dépréciée. Il vaut mieux utiliser `epub.open_epub()` à la place, qui prend les mêmes paramètres, mais évite tout potentiel conflit avec la fonction python *open()*.
— Ajout d'une note sur la version et la compatibilité avec les versions de Python dans la documentation.

Il n'est plus possible d'utiliser la méthode *epub.EpubFile.read* avec un `ManifestItem` directement.

La fonction *epub.open()* est dépréciée et émet un warning. Il faut utiliser la fonction `epub.open_epub()` à la place.

Bug fix

Les bugs suivants ont été résolus :
— Issue #2 : *"Documentation pub on tutorial page."* La documentation ne fait plus de références à la méthode *read* dans ses exemples.
— Issue #3 : *"_parse_xml_metadata should not choke on absent firstChild"* Une fonction a été ajoutée pour traiter la récupération de texte dans un noeud xml, et cette fonction est suffisament intelligente pour ne pas planter au premier texte manquant. Voir aussi `epub.utils.get_node_text()`.

Version 0.4.0

Cette nouvelle version propose plusieurs petites améliorations, ainsi qu'une nouvelle fonctionnalité majeure : le mode écriture.

L'API n'est plus tout à fait la même, notamment les méthodes pour lire les fichiers ont changé un peu. Au vu des grands changements introduits par cette version, elle **n'est pas compatible** avec les versions précédentes.

De plus, cette version n'est compatible qu'avec Python 2.7. Il est prévu de supporter Python 2.6 dans une prochaine version.
— La fonction `epub.open()` accepte un second paramètre *mode* pour choisir d'ouvrir le fichier en lecture seule ou en écriture.
— La méthode `epub.EpubFile.read()` devient `epub.EpubFile.read_item()`. Cette méthode possède le même fonctionnement que l'ancienne, qui reprend sa fonctionnalité native de la classe *zipfile.ZipFile*.
— La classe `epub.opf.Manifest` étend la classe `collection.OrderedDict` et plus la type *dict*. Ce changement n'est pas compatible avec une autre version que Python 2.7, mais un backport sera proposé prochainement.
— Correction de divers bug, notamment sur le chargement des méta-données.
— Une meilleure documentation du module `epub.opf` et `epub.ncx`.
— Une couverture à 100% des tests unitaires.
— Ajout de ce fichier de changelog.

CHAPITRE 6

Introduction

Connaissez-vous le format de livre numérique appelé “Epub” ? Il s’agit d’un format ouvert, proposé par l’IDPF, qui le présente ainsi :

> EPUB is the distribution and interchange format standard for digital publications and documents based on Web Standards. EPUB defines a means of representing, packaging and encoding structured and semantically enhanced Web content — including XHTML, CSS, SVG, images, and other resources — for distribution in a single-file format.
>
> EPUB allows publishers to produce and send a single digital publication file through distribution and offers consumers interoperability between software/hardware for unencrypted reflowable digital books and other publications.

En d’autre terme : c’est un format qui permet de rassembler dans un seul fichier un ensemble de fichiers de type “html&css”, pour publier du contenu - du genre, un livre numérique.

Dans le monde python, si vous cherchez bien, vous trouverez des solutions tout à fait pertinentes d’applications pour lire et éditer des fichiers epub.

Cependant, il s’agit beaucoup de solutions spécifiques (et en outre, très peu de bibliothèques), d’où l’existence de cette bibliothèque qui permet d’ouvrir (en lecture seule pour le moment) des fichiers au format epub (dans la version 2 de la spécification, pour le moment).

Voir aussi : le site l’IDPF et la spécification Epub 2.

Avertissement : Pour le moment, cette bibliothèque est en phase de développement. Chaque release effectuée sur pypi est testée unitairement (ce qui évite en théorie la présence de bugs), mais l’API n’est pas dans une forme stable. Des changements et évolutions sont notamment à prévoir sur la façon d’accéder aux fichiers contenus dans l’archive epub.

En outre il est vivement déconseillé d’utiliser la version de développement à partir du repository mercurial, ce derniers étant tout sauf stable. Vous êtes par contre vivement encouragés à l’utiliser pour détecter des bugs, proposer des améliorations, et/ou remonter toutes les incohérences et/ou maladresses présentes dans le code.

Les bonnes volontés et les remarques sont **toutes** bonnes à prendre.

CHAPITRE 7

Licence

La licence choisie pour cette bibliothèque est la LGPL.

CHAPITRE 8

Installation

Disponible sur pypi, vous pouvez installer Python Epub via la commande “pip” :

```
pip install epub
```

Sinon, vous pouvez obtenir la dernière version des sources via mercurial :

```
hg clone https://bitbucket.org/exirel/epub
cd epub
python setup.py install
```

CHAPITRE 9

Version et compatibilité

Le module *epub* est disponible en version stable *0.5.0* depuis le *14 Octobre 2012*.

Cette version est **compatible et testée** avec Python 2.7 et Python 3.2.

Cette version est **potentiellement compatible** avec Python 2.6, mais par manque de tests unitaires le mainteneur du module ne peut pas garantir de compatibilité.

Cette version **n'est pas compatible** avec des versions de Python antérieures à la version 2.6.

CHAPITRE 10

Utilisation

Le cas d'utilisation le plus simple est représenté par le code suivant :

```
import epub

book = epub.open_epub('path/to/my/book.epub')

for item in book.opf.manifest.values():
    # read the content
    data = book.read_item(item)
```

Bien entendu, ce n'est qu'un exemple, très incomplet qui plus est, de ce que vous pouvez faire.

Cette bibliothèque ayant pour ambition de suivre les spécifications du format epub, certains éléments pourront paraître obscurs.

Rassurez-vous, cette documentation est là pour vous apporter le plus possible de réponses à vos interrogations.

CHAPITRE 11

Indices and tables

— genindex
— modindex
— search

Index des modules Python

e

epub, 1
epub.ncx, 15
epub.opf, 7
epub.utils, 23

Index

Symbols

__init__() (méthode epub.EpubFile), 3

A

add_contributor() (méthode epub.opf.Metadata), 11
add_creator() (méthode epub.opf.Metadata), 11
add_date() (méthode epub.opf.Metadata), 12
add_identifier() (méthode epub.opf.Metadata), 12
add_info() (méthode epub.ncx.NavList), 20
add_info() (méthode epub.ncx.NavMap), 18
add_info() (méthode epub.ncx.PageList), 19
add_item() (méthode epub.EpubFile), 4
add_item() (méthode epub.opf.Manifest), 12
add_itemref() (méthode epub.opf.Spine), 14
add_label() (méthode epub.ncx.NavList), 20
add_label() (méthode epub.ncx.NavMap), 18
add_label() (méthode epub.ncx.NavPoint), 18
add_label() (méthode epub.ncx.NavTarget), 21
add_label() (méthode epub.ncx.PageList), 19
add_label() (méthode epub.ncx.PageTarget), 20
add_language() (méthode epub.opf.Metadata), 12
add_meta() (méthode epub.opf.Metadata), 12
add_nav_list() (méthode epub.ncx.Ncx), 17
add_point() (méthode epub.ncx.NavMap), 18
add_point() (méthode epub.ncx.NavPoint), 19
add_reference() (méthode epub.opf.Guide), 14
add_subject() (méthode epub.opf.Metadata), 11
add_target() (méthode epub.ncx.NavList), 21
add_target() (méthode epub.ncx.PageList), 19
add_title() (méthode epub.opf.Metadata), 11
append() (méthode epub.opf.Guide), 14
append() (méthode epub.opf.Manifest), 13
as_xml_document() (méthode epub.ncx.NavList), 21
as_xml_document() (méthode epub.ncx.NavMap), 18
as_xml_document() (méthode epub.ncx.NavPoint), 19
as_xml_document() (méthode epub.ncx.NavTarget), 21
as_xml_document() (méthode epub.ncx.Ncx), 17
as_xml_document() (méthode epub.ncx.PageList), 19
as_xml_document() (méthode epub.ncx.PageTarget), 20
as_xml_element() (méthode epub.opf.Manifest), 13
as_xml_element() (méthode epub.opf.ManifestItem), 14
as_xml_element() (méthode epub.opf.Metadata), 12
authors (attribut epub.ncx.Ncx), 17

B

Book (classe dans epub), 5

C

check_mode_write() (méthode epub.EpubFile), 4
class_name (attribut epub.ncx.NavList), 20
class_name (attribut epub.ncx.NavPoint), 18
class_name (attribut epub.ncx.NavTarget), 21
class_name (attribut epub.ncx.PageList), 19
class_name (attribut epub.ncx.PageTarget), 20
close() (méthode epub.EpubFile), 4
contributors (attribut epub.opf.Metadata), 11
coverage (attribut epub.opf.Metadata), 11
creators (attribut epub.opf.Metadata), 10

D

dates (attribut epub.opf.Metadata), 11
dc_type (attribut epub.opf.Metadata), 11
depth (attribut epub.ncx.Ncx), 17
description (attribut epub.opf.Metadata), 11

E

epub (module), 1
epub.ncx (module), 15
epub.opf (module), 7
epub.utils (module), 23
EpubFile (classe dans epub), 3
extract_item() (méthode epub.EpubFile), 4

F

fallback (attribut epub.opf.ManifestItem), 13
fallback_style (attribut epub.opf.ManifestItem), 13
format (attribut epub.opf.Metadata), 11

G

generator (attribut epub.ncx.Ncx), 17
get_isbn() (méthode epub.opf.Metadata), 12
get_item() (méthode epub.EpubFile), 4
get_item_by_href() (méthode epub.EpubFile), 4
get_node_text() (dans le module epub.utils), 23
get_urlpath_part() (dans le module epub.utils), 23
guide (attribut epub.opf.Opf), 10
Guide (classe dans epub.opf), 14

H

href (attribut epub.opf.ManifestItem), 13

I

identifier (attribut epub.ncx.NavList), 20
identifier (attribut epub.ncx.NavMap), 18
identifier (attribut epub.ncx.NavPoint), 18
identifier (attribut epub.ncx.NavTarget), 21
identifier (attribut epub.ncx.PageList), 19
identifier (attribut epub.ncx.PageTarget), 19
identifier (attribut epub.opf.ManifestItem), 13
identifiers (attribut epub.opf.Metadata), 11
infos (attribut epub.ncx.NavList), 20
infos (attribut epub.ncx.NavMap), 18
infos (attribut epub.ncx.PageList), 19
itemrefs (attribut epub.opf.Spine), 14

L

labels (attribut epub.ncx.NavList), 20
labels (attribut epub.ncx.NavMap), 18
labels (attribut epub.ncx.NavPoint), 18
labels (attribut epub.ncx.NavTarget), 21
labels (attribut epub.ncx.PageList), 19
labels (attribut epub.ncx.PageTarget), 20
lang (attribut epub.ncx.Ncx), 17
languages (attribut epub.opf.Metadata), 11

M

manifest (attribut epub.opf.Opf), 10
Manifest (classe dans epub.opf), 12
ManifestItem (classe dans epub.opf), 13
max_page_number (attribut epub.ncx.Ncx), 17
media_types (attribut epub.opf.ManifestItem), 13
metadata (attribut epub.opf.Opf), 10
Metadata (classe dans epub.opf), 10
metas (attribut epub.opf.Metadata), 11

N

nav_lists (attribut epub.ncx.Ncx), 17
nav_map (attribut epub.ncx.Ncx), 17
nav_point (attribut epub.ncx.NavMap), 18
nav_point (attribut epub.ncx.NavPoint), 18
nav_target (attribut epub.ncx.NavList), 20
NavList (classe dans epub.ncx), 20
NavMap (classe dans epub.ncx), 18
NavPoint (classe dans epub.ncx), 18
NavTarget (classe dans epub.ncx), 21
Ncx (classe dans epub.ncx), 17

O

open_epub() (dans le module epub), 3
opf (attribut epub.EpubFile), 3
Opf (classe dans epub.opf), 10
opf_path (attribut epub.EpubFile), 3

P

page_list (attribut epub.ncx.Ncx), 17
page_target (attribut epub.ncx.PageList), 19
PageList (classe dans epub.ncx), 19
PageTarget (classe dans epub.ncx), 19
parse_ncx() (dans le module epub.ncx), 17
parse_opf() (dans le module epub.opf), 10
play_order (attribut epub.ncx.NavPoint), 18
play_order (attribut epub.ncx.NavTarget), 21
play_order (attribut epub.ncx.PageTarget), 20
publisher (attribut epub.opf.Metadata), 11

R

read_item() (méthode epub.EpubFile), 4
references (attribut epub.opf.Guide), 14
relation (attribut epub.opf.Metadata), 11
required_modules (attribut epub.opf.ManifestItem), 13
required_namespace (attribut epub.opf.ManifestItem), 13
right (attribut epub.opf.Metadata), 11

S

source (attribut epub.opf.Metadata), 11
spine (attribut epub.opf.Opf), 10
Spine (classe dans epub.opf), 14
src (attribut epub.ncx.NavPoint), 18
src (attribut epub.ncx.NavTarget), 21
src (attribut epub.ncx.PageTarget), 20
subjects (attribut epub.opf.Metadata), 10

T

target_type (attribut epub.ncx.PageTarget), 20
title (attribut epub.ncx.Ncx), 17
titles (attribut epub.opf.Metadata), 10
toc (attribut epub.EpubFile), 3
toc (attribut epub.opf.Spine), 14
total_page_count (attribut epub.ncx.Ncx), 17

U

uid (attribut epub.EpubFile), 3
uid (attribut epub.ncx.Ncx), 17
uid_id (attribut epub.opf.Opf), 10

V

value (attribut epub.ncx.NavTarget), 21
value (attribut epub.ncx.PageTarget), 20
version (attribut epub.ncx.Ncx), 17
version (attribut epub.opf.Opf), 10

X

xmlns (attribut epub.ncx.Ncx), 17
xmlns (attribut epub.opf.Opf), 10