libacbf

Grafcube

Nov 09, 2021

CONTENTS:

This is a Python library to read and edit ACBF formatted comic book files and archives.

See ACBF Specifications for information on the XML schema.

CHAPTER

ONE

INSTALLATION AND DEPENDENCIES

1.1 Installation

```
>>> pip install libacbf
```

1.2 Dependencies

- libacbf uses `python-magic` (PyPI, GitHub) which has its own dependencies. Make sure you install those.
- It uses `rarfile` (PyPI, GitHub, Docs) to read Rar archived books and it has its own dependencies listed here. Install those if you want to read RAR archives. Editing a RAR archive is not possible.

CHAPTER

TWO

QUICK TUTORIAL

2.1 Read a book

Use it with a context manager:

```
from libacbf import ACBFBook

with ACBFBook("path/to/book.cbz") as book:
    # Read English title of book
    title = book.book_info.book_titles["en"]
```

Alternatively just open and close it:

```
from libacbf import ACBFBook

book = ACBFBook("path/to/book.cbz")
# Read English title of book
title = book.book_info.book_titles["en"]
book.close()
```

You can read plain `.acbf` XML formatted files, Zip archives, 7Zip archives, Tar archives or RAR archives.

2.1.1 Metadata

It is recommended that you skim through the API Reference at least once. Specifically ACBFBook, Book Info, Publisher Info and Document Info to see all the metadata you can read.

```
book.book_info  # Read metadata from the book info section
book.publish_info  # Read metadata from the publisher info section
book.document_info  # Read metadata from the document info section
```

2.1.2 Body

(Read ACBFBody in the API Reference)

It contains a list of pages that you can read info from.

```
for page in book.body.pages:
    page  # A page object with information
    page.image  # Get the image data from its source.
```

2.1.3 Data

(Read ACBFData in the API Reference)

Files can be embedded within the plain XML book. You can use this to list and read the embedded data.

```
book.data.list_files()
```

2.1.4 Styles

(Read Styles in the API Reference)

Stylesheets can be used with an ACBF formatted book. Styles can be embedded in a style tag, in the data section or be a reference to another file either in an archive or a path on disk.

```
book.styles.list_styles()
```

2.2 Edit/Create Books

(See ACBFBook for detailed information)

You can use different modes to open books.

```
from libacbf import ACBFBook

# Edit an existing file.
with ACBFBook("path/to/book.cbz", 'a') as book:
    # Edit the English title of the book
    book.book_info_book_titles["en"] = "New title"
```

You can create new files with other modes. `'w'` will create a new file at the given path. If a file already exists it will be overwritten so be careful. `'x'` will also create a new file but raises `FileExistsError` if a file already exists.

```
from libacbf import ACBFBook

# Creates a new book. Overwrites if a file already exists.
with ACBFBook("path/to/file.cbz", 'w') as book:
    # Set the English title of the new book
    book.book_info.book_titles["en"] = "Newly created book"

# Creates a new book. Raises an exception if a file already exists.
```

(continues on next page)

(continued from previous page)

```
with ACBFBook("path/to/file.cbz", 'x') as book:
    # Set the English title of the new book
    book.book_info.book_titles["en"] = "Newly created book"
```

By default, the book will be a Zip archive. You can override this with the nullable `archive_type` parameter. Accepted values can be found in the API Reference. This parameter is ignored when using `'r'` or `'a'` mode.

Passing `None` creates a plain text XML formatted book. You can convert it to an archive later if you want using `ACBFBook.make_archive(...)`. You cannot create Rar archived books.

> **Warning:** You will lose information like compression level etc. when you edit and/or create a book as it re-archives the book using the default values for each archive type. Information on how to avoid this is available further below.

```
from libacbf import ACBFBook

# Creates a Zip archived book
with ACBFBook("path/to/file.cbz", 'w', archive_type="Zip") as book:
    book.book_info.book_titles["en"] = "CBZ book"

# Creates a 7Zip archived book
with ACBFBook("path/to/file.cb7", 'w', archive_type="SevenZip") as book:
    book.book_info.book_titles["en"] = "CB7 book"

# Creates a Tar archived book
with ACBFBook("path/to/file.cbt", 'w', archive_type="Tar") as book:
    book.book_info.book_titles["en"] = "CBT book"

# Creates a plain XML book
with ACBFBook("path/to/file.acbf", 'w', archive_type=None) as book:
    book.book_info.book_titles["en"] = "ACBF book"

    # This can be converted to an archive later
    # An exception is raised if the book is already an archive

    # Convert to CBZ
    book.make_archive()
    -- OR --
    book.make_archive("Zip")

    # Convert to CB7
    book.make_archive("SevenZip")

    # Convert to CBT
    book.make_archive("Tar")
```

2.3 Edit Book Data

`ACBFBook.book_info.genres` is a dictionary with keys being enum values. You can edit it by doing this:

```
from libacbf import ACBFBook
from libacbf.constants import Genres

with ACBFBook("path/to/file.cbz", 'a') as book:
    # Get the match value of the genre if it exists
    match = book.book_info.genres[Genres.other]

    # Add a new genre if it doesn't already exist and set no match value
    book.book_info.genres[Genres.humor] = None

    # Add a genre and set its match value to 90%
    book.book_info.genres[Genres.manga] = 90
```

Or you could do this:

```
from libacbf import ACBFBook

with ACBFBook("path/to/file.cbz", 'w') as book:
    # Get the match value of the genre if it exists
    match = book.book_info.get_genre_match("other")

    # Add a new genre if it doesn't already exist and set no match value
    book.book_info.edit_genre("humor")

    # Add a genre and set its match value to 90%
    book.book_info.edit_genre("manga", 90)
```

Similarly you can add an author object to `ACBFBook.document_info.authors`.

Importing Author:

```
from libacbf import ACBFBook
from libacbf.metadata import Author

with ACBFBook("path/to/file.cbz", 'w') as book:
    book.document_info.authors.append( Author("Nickname") )
```

Directly:

```
from libacbf import ACBFBook

with ACBFBook("path/to/file.cbz", 'w') as book:
    book.document_info.add_author("Nickname")
```

There are many functions available that simplify editing, allowing you to edit information without having to import additional classes.

2.4 Adding Pages

Adding pages may take one or two steps.

First let's append a page to the book.

```
from libacbf import ACBFBook

with ACBFBook("path/to/file.cbz", 'w') as book:
    book.body.append_page("page1.png")
```

The page is now referencing an image relative to the root of the Zip archive. If the archive already has this image then you're done. If it doesn't, you have to write it to the archive.

```
# ... continued
book.data.add_data("path/to/image/page1.png")
```

This will write the image stored on disk into the archive with the path `"page1.png"` relative to the root of the archive. There is more information on what you can do with this function in the API reference.

2.5 How to avoid losing the original compression of an archive

Regardless of whether you open a book in `'w'`, `'a'` or `'x'` mode, it is saved with the default options of its archive type. So for example, if your CBZ book uses `ZIP_DEFLATE` compression, opening it will extract and re-archive it as `ZIP_STORED` because that is the default. Books opened with `'r'` do not affect the original file in any way.

To get around this, you can manage the archive manually. Image references in pages are relative to the root of the archive and the `.acbf` file must also be at the root of the archive. If you extract the contents of the archive to a directory, the image references will be relative to the ACBF file and it will still retrieve the correct image. You can then edit the book as usual. The only difference would be that writing a file to the archive means copying the file into the extracted directory.

After you're done you can archive the contents of the directory with the settings you want and read the archived book normally.

CHAPTER

THREE

API REFERENCE

3.1 Main module

3.1.1 get_book_template()

libacbf.**get_book_template**(*ns: Optional[str] = None*) → str

Get the bare minimum XML required to create an ACBF book.

Returns XML string template.

Return type str

3.1.2 ACBFBook

class libacbf.**ACBFBook**(*file: Union[str, pathlib.Path, IO]*, *mode: Literal['r', 'w', 'a', 'x'] = 'r'*, *archive_type: Optional[str] = 'Zip'*)

Bases: `object`

Base class for reading ACBF ebooks.

Parameters

- **file** (*str | Path | IO*) – Path or file object to write ACBF book to. May be absolute or relative.
- **mode** (*'r' | 'w' | 'a' | 'x', default='r'*) – The mode to open the file in. Defaults to read-only mode.

  r Read only mode. No editing is possible. Can read ACBF, Zip, 7Zip, Tar and Rar formatted books.

  w Overwrite file with new file. Raises exception for Rar archive types.

  a Edit the book without truncating. Raises exception for Rar archive types.

  x Exclusive write to file. Raises `FileExists` exception if file already exists. Only works for file paths. Raises exception for Rar archive types.
- **archive_type** (*str | None, default="Zip"*) – The type of ACBF book that the file is. If `None` Then creates a plain XML book. Otherwise creates archive of format. Accepted string values are listed at *ArchiveTypes*.

> **Warning:** You do not have to specify the type of archive unless you are creating a new one. The correct type will be determined regardless of this parameter's value. Use this when you want to create a new book.

Raises

- *EditRARArchiveError* – Raised if `mode` parameter is not `'r'` but file is a Rar archive.
- *InvalidBook* – Raised if the XML does not match ACBF schema or if archive does not contain ACBF file.

See also:

ACBF Specifications.

Notes

Archive formats use the defaults of each type like compression level etc. Manage the archives yourself if you want to change this. Image refs that are relative paths check within the archive if the book is an archive. Otherwise it checks relative to the '.acbf' file. So you can simply use a directory to manage the book and archive it with your own settings when you are done.

Examples

A book object can be opened, read and then closed.

```
from libacbf import ACBFBook

book = ACBFBook("path/to/file.cbz")
# Read data from book
book.close()
```

`ACBFBook` is also a context manager and can be used in with statements.

```
from libacbf import ACBFBook

with ACBFBook("path/to/file.cbz") as book:
    # Read data from book
```

You can pass a `BytesIO` object. Keep in mind that you cannot use `mode='x'` in this case.

```
import io
from libacbf import ACBFBook

file = io.BytesIO()

with ACBFBook(file, 'w') as book:
    # Write data to book
```

book_info

See `BookInfo` for more information.

Type *BookInfo*

publisher_info
See `PublishInfo` for more information.

Type *PublishInfo*

document_info
See `DocumentInfo` for more information.

Type *DocumentInfo*

body
See `ACBFBody` for more information.

Type *ACBFBody*

data
See `ACBFData` for more information.

Type *ACBFData*

references
A dictionary that contains a list of particular references that occur inside the main document body. Keys are unique reference ids and values are dictionaries that contain a `'_'` key with text.

```
{
    "ref_id_001": {
        "_": "This is a reference."
    }
    "ref_id_002": {
        "_": "This is another reference."
    }
}
```

`'_'` can contain special tags for formatting. For more information and a full list, see *TextArea.text*.

Type dict

styles
See `Styles` for more information.

Type *Styles*

archive
Can be used to read archive directly if file is not plain ACBF. Use this if you want to read exactly what files the book contains but try to avoid directly writing files through `ArchiveReader`.

Type *ArchiveReader* | None

get_acbf_xml() → str
Get the XML tree of the ACBF book.

Returns The XML content of the ACBF book.

Return type str

make_archive(*archive_type: str = 'Zip'*)
Convert a plain ACBF XML book to an archive format.

Parameters **archive_type** (*str, default="Zip"*) – The type of archive to create. Allowed values are listed at *ArchiveTypes*.

Raises **AttributeError (Book is already an archive of type {archive.type}.)** – Raised when book is already an archive.

`close()`
Saves and closes the book and closes the archive if it exists. Metadata and embedded data can still be read. Use `ACBFBook.is_open` to check if file is open.

3.1.3 Book Info

class libacbf.libacbf.**BookInfo**(*book:* libacbf.libacbf.ACBFBook)
Bases: `object`

Metadata about the book itself.

See also:

Book-Info section.

`authors`
A list of *Author* objects.

Type List[*Author*]

`book_title`
A dictionary with standard language codes as keys and titles as values. Key is `'_'` if no language is defined.

```
{
    "_": "book title without language",
    "en": "English title",
    "en_GB": "English (UK) title",
    "en_US": "English (US) title"
}
```

Type Dict[str, str]

`genres`
A dictionary with keys being a value from *constants.Genres* Enum and values being integers with the match value or `None`. See `get_match()`.

Type Dict[*Genres*, int | None]

`annotations`
A short summary describing the book.

It is a dictionary with keys being standard language codes or `'_'` if no language is defined and values being multiline strings.

Type Dict[str, str]

`coverpage`
It is the same as *body.Page* except it does not have *body.Page.title*, *body.Page.bgcolor* and *body.Page.transition*.

Type *Page*

`languages`
It represents all *body.TextLayer* objects of the book.

A list of *LanguageLayer* objects.

Type List[*LanguageLayer*], optional

`characters`
List of (main) characters that appear in the book.

Type List[str], optional

keywords
For use by search engines.

A dictionary with keys as standard language codes or `'_'` if no language is defined. Values are a set of lowercase keywords.

Type Dict[str, Set[str]], optional

series
Contains the sequence and number if particular comic book is part of a series.

A dictionary with keys as the title of the series and values as *Series* objects.

Type Dict[str, *Series*], optional

content_rating
Content rating of the book based on age appropriateness and trigger warnings.

It is a dictionary with the keys being the rating system and values being the rating.

```
{
    "Age Rating": "16+",
    "DC Comics rating system": "T+",
    "Marvel Comics rating system": "PARENTAL ADVISORY"
}
```

Type Dict[str, str], optional

database_ref
References to a record in a comic book database (eg: GCD, MAL).

A list of *DBRef* objects.

Type List[*DBRef*], optional

add_author(**names: str*, *first_name=None*, *last_name=None*, *nickname=None*) → *libacbf.metadata.Author*
Add an Author to the book info. Usage is the same as *Author*.

Returns The created Author object.

Return type *Author*

get_genre_match(*genre: str*) → int
Get match value of genre by string.

edit_genre(*genre: str*, *match: Optional[int] = '_'*)
Edit a genre by string. Add it if it doesn't exist.

Parameters

- **genre** (*str*) – See *constants.Genres* enum for a list of possible values.
- **match** (*int | None, optional*) – Set the match percentage of the genre. If None, removes the match value.

pop_genre(*genre: str*) → Optional[int]
Pop a genre by string.

Returns The match value of the genre.

Return type int | None

add_language(*lang: str*, *show: bool*)
Add a language layer to the book. Usage is the same as *LanguageLayer*.

add_series(*title: str*, *sequence: str*, *volume: Optional[str] = None*)
Add a series that the book belongs to. `title` is the key and usage for value is the same as *Series*.

add_dbref(*dbname: str*, *ref: str*, *type: Optional[str] = None*)
Add a database reference to the book. Usage is the same as *DBRef*.

3.1.4 Publisher Info

class libacbf.libacbf.**PublishInfo**(*book:* libacbf.libacbf.ACBFBook)
Bases: `object`

Metadata about the book's publisher.

See also:

Publish-Info section.

publisher
Name of the publisher.

Type str

publish_date
Date when the book was published as a human readable string.

Type str

publish_date_value
Date when the book was published.

Type datetime.date, optional

publish_city
City where the book was published.

Type str, optional

isbn
International Standard Book Number.

Type str, optional

license
The license that the book is under.

Type str, optional

set_date(*date: Union[str, datetime.date]*, *include_date: bool = True*)
Edit the date the book was published.

Parameters

- **date** (*str | datetime.date*) – Date to set to.
- **include_date** (*bool, default=True*) – Whether to also set *publish_date_value*. Passing `False` will set it to `None`.

3.1.5 Document Info

class libacbf.libacbf.**DocumentInfo**(*book:* libacbf.libacbf.ACBFBook)

Bases: object

Metadata about the ACBF file itself.

See also:

Document-Info section.

authors
Authors of the ACBF file as a list of *Author* objects.

Type List[*Author*]

creation_date
Date when the ACBF file was created as a human readable string.

Type str

creation_date_value
Date when the ACBF file was created.

Type datetime.date, optional

source
A multiline string with information if this book is a derivative of another work. May contain URL and other source descriptions.

Type str, optional

document_id
Unique Document ID. Used to distinctly define ACBF files for cataloguing.

Type str, optional

document_version
Version of ACBF file.

Type str, optional

document_history
Change history of the ACBF file with change information in a list of strings.

Type List[str], optional

add_author(**names: str*, *first_name=None*, *last_name=None*, *nickname=None*) → *libacbf.metadata.Author*
Add an Author to the document info. Usage is the same as *Author*.

Returns The created Author object.

Return type *Author*

set_date(*date: Union[str, datetime.date]*, *include_date: bool = True*)
Edit the date the ACBF file was created.

Parameters

- **date** (*str | datetime.date*) – Date to set to.
- **include_date** (*bool, default=True*) – Whether to also set *creation_date_value*. Passing False will set it to None.

3.1.6 Body

class libacbf.libacbf.**ACBFBody**(*book:* libacbf.libacbf.ACBFBook)
Bases: object

Body section contains the definition of individual book pages and text layers, frames and jumps inside those pages.

See also:

Body Section Definition.

pages
A list of *Page* objects in the order they should be displayed in.

Type List[*Page*]

bgcolor
Defines a background colour for the whole book. Can be overridden by bgcolor in pages, text layers, text areas and frames.

Type str, optional

insert_page(*index: int*, *image_ref: str*) → *libacbf.body.Page*
Insert a new Page object at the index.

Parameters

- **index** (*int*) – Index of new page.
- **image_ref** (*str*) – Value to set for the image reference. See *Page.image_ref* for information on how to format it.

Returns The created Page object.

Return type *Page*

append_page(*image_ref: str*) → *libacbf.body.Page*
Append a new Page object to the body.

Parameters **image_ref** (*str*) – Value to set for the image reference. See *Page.image_ref* for information on how to format it.

Returns The created Page object.

Return type *Page*

3.1.7 Data

class libacbf.libacbf.**ACBFData**(*book:* libacbf.libacbf.ACBFBook)
Bases: object

Get any binary data embedded in the ACBF file or write data to archive or embed data in ACBF.

See also:

Data Section Definition.

Returns A file as a *BookData* object.

Return type *BookData*

Raises **FileNotFoundError** – Raised if file is not found embedded in the ACBF file.

Examples

To get a file embedded in the ACBF file:

```
from libacbf import ACBFBook

with ACBFBook("path/to/book.cbz") as book:
    image = book.data["image.png"]
    font = book.data["font.ttf"]
```

list_files() → Set[str]
Returns a list of all the names of the files embedded in the ACBF file. May be images, fonts etc.

Returns A set of file names.

Return type Set[str]

add_data(*target: Union[str, pathlib.Path, bytes]*, *name: str = None*, *embed: bool = False*)
Add or embed data into the book.

Parameters

- **target** (*str | Path | bytes*) – Path to file to be added or data as bytes.
- **name** (*str, optional*) – Name to assign to file after writing. Defaults to name part of target. Required if `target` is bytes.
- **embed** (*bool, default=False*) – Whether to embed the file in the ACBF XML. Cannot be `False` if book is not an archive type. Use *ACBFBook.make_archive(...)* to convert the book to an archive.

remove_data(*target: Union[str, pathlib.Path]*, *embed: bool = False*)
Remove file at target in the archive. If `embed` is true, removes from embedded files.

Parameters

- **target** (*str | Path*) – Path to file in archive or id of embedded file.
- **embed** (*bool, default=False*) – Whether to check for file in archive or embedded in ACBF XML. Must be true if book is plain ACBF XML.

3.1.8 Styles

class libacbf.libacbf.**Styles**(*book:* libacbf.libacbf.ACBFBook)
Bases: `object`

Stylesheets to be used in the book.

See also:

Stylesheet Declaration.

Returns Stylesheet data.

Return type bytes

Examples

To get stylesheets

```
from libacbf import ACBFBook

with ACBFBook("path/to/book.cbz") as book:
    style1 = book.styles["style1.css"]  # Style referenced at the top of the ACBF␣
↪XML.
    embedded_style = book.styles['_']  # Returns the stylesheet embedded in ACBF␣
↪XML.
```

types
A dictionary with keys being the style name (or `'_'`) and values being the type or `None` if not specified.

Type Dict[str, str | None]

list_styles() → Set[str]
All the stylesheets referenced by the ACBF XML.

Returns Referenced stylesheets.

Return type Set[str]

edit_style(*stylesheet: Union[str, pathlib.Path, bytes]*, *style_name: str = None*, *type: str = 'text/css'*, *embed: bool = False*)
Writes or overwrites file in book with referenced stylesheet.

Parameters

- **stylesheet** (*str | Path | bytes*) – Path to stylesheet or stylesheet as bytes.
- **style_name** (*str, optional*) – Name of stylesheet after being written. Defaults to name part of `stylesheet_ref`. If it is `'_'`, writes stylesheet to style tag of ACBF XML. Required if `stylesheet` is bytes.
- **type** (*str, default="text/css"*) – Mime Type of stylesheet. Defaults to CSS but can be others (like SASS).
- **embed** (*bool, default=False*) – Whether to embed stylesheet in the data section of the book. This is ignored if `style_name` is `'_'`. Must be True if book is plain ACBF XML. Use *ACBFBook.make_archive(...)* to convert the book to an archive.

remove_style(*style_name: str*, *embedded: bool = False*)
Remove stylesheet from book.

Parameters

- **style_name** (*str*) – Stylesheet to remove. If it is `'_'`, remove embedded stylesheet.
- **embedded** (*bool, default=False*) – Remove style from embedded data of book. Ignored if style_name is `'_'`. Must be False if book is plain ACBF XML.

3.2 Metadata

3.2.1 Author

class libacbf.metadata.**Author**(**names: str*, *first_name=None*, *last_name=None*, *nickname=None*)

Bases: `object`

Defines an author of the comic book. An author must at least have a nickname not be `None` or have both first name and last name not be `None`.

See also:

Author specifications.

Examples

An `Author` object can be created with either a nickname, a first and last name or both.

```
from libacbf import ACBFBook

with ACBFBook("path/to/book.cbz", 'w') as book:
    au = Author("Hugh", "Mann")
    # au.first_name == "Hugh"
    # au.last_name == "Mann"

    au = Author("NotAPlatypus")
    # au.nickname == "NotAPlatypus"

    au = Author("Hugh", "Mann", "NotAPlatypus")
    # au.first_name == "Hugh"
    # au.last_name == "Mann"
    # au.nickname == "NotAPlatypus"
```

This is also possible.

```
au = Author(first_name="Hugh", last_name="Mann", nickname="NotAPlatypus")
```

first_name
Author's first name.

Type str

last_name
Author's last name.

Type str

nickname
Author's nickname.

Type str

middle_name
Author's middle name.

Type str, optional

home_page
Author's website.

Type str, optional

email
Author's email address.

Type str, optional

property activity: Optional[*libacbf.constants.AuthorActivities*]
Defines the activity that a particular author carried out on the comic book. Allowed values are defined in *AuthorActivities*.

Returns A value from *AuthorActivities* enum or `None` if not defined.

Return type *AuthorActivities*(Enum) | None

property lang: Optional[str]
Defines the language that the author worked in.

Returns Returns a standard language code or `None` if not defined.

Return type str | None

copy()
Returns a copy of this object.

3.2.2 LanguageLayer

class libacbf.metadata.**LanguageLayer**(*lang: str*, *show: bool*)
Bases: `object`

Used by *ACBFBook.book_info.languages*.

See also:

Book Info Languages specifications.

lang
Language of the layer as a standard language code.

Type str

show
Whether the layer is drawn.

Type bool

3.2.3 Series

class libacbf.metadata.**Series**(*sequence: str*, *volume: Optional[str] = None*)
Bases: `object`

Used by *ACBFBook.book_info.series*.

See also:

Book Info Sequence specifications.

sequence
The book's position/entry in the series.

Type str

volume
The volume that the book belongs to.

Type str, optional

3.2.4 DBRef

class libacbf.metadata.**DBRef**(*dbname: str*, *ref: str*, *type: Optional[str] = None*)
Bases: object

Used by *ACBFBook.book_info.database_ref*.

See also:

Book Info DatabaseRef specifications.

dbname
Name of the database.

Type str

reference
Reference of the book in the database.

Type str

type
Type of the given reference such as URL, ID etc.

Type str, optional

3.3 Body

3.3.1 Page

class libacbf.body.**Page**(*image_ref: str*, *book: ACBFBook*, *coverpage: bool = False*)
Bases: object

A page in the book.

See also:

Page Definition.

image_ref
Reference to the image file. May be embedded in the ACBF file, in the ACBF archive, in an external archive, a local path or a URL.

There are several ways to format it to read data:

Reference to a file embedded in *ACBFBook.data*:

- "#page1.jpg"

Reference to a file on disk:

- "/path/to/file/page1.jpg"
- "C:\path\to\file\page1.jpg"

- “file:///path/to/file/page1.jpg“
- “file://C:\path\to\file\page1.jpg“

Path to a file in the book’s archive or relative path to file on disk if book is a plain ACBF XML:

- “page1.jpg“
- “images/page1.jpg“

Reference to file in an archive:

- “zip:path/to/archive.zip!/path/to/file/page1.jpg“

URL address containing the image:

- “https://example.com/book1/images/page1.jpg“

Type str

ref_type
A value from *ImageRefType* indicating the type of reference in *Page.image_ref*.

Type *ImageRefType*(Enum)

text_layers
A dictionary with keys being the language of the text layer and values being *TextLayer* objects.

Type Dict[str, *TextLayer*]

frames
A list of *Frame* objects in order of appearance.

Type List[*Frame*]

jumps
A list of *Jump* objects.

Type List[*Jump*]

Warning: The attributes `title`, `bgcolor` and `transition` are not available on *ACBFBook.book_info.coverpage*.

title
It is used to define beginning of chapters, sections of the book and can be used to create a table of contents. Keys are standard language codes or `'_'` if not defined. Values are titles as string.

Type Dict[str, str], optional

bgcolor
Defines the background colour for the page. Inherits from *ACBFBody.bgcolor* if `None`.

Type str, optional

transition
Defines the type of transition from the previous page to this one. Allowed values are in *PageTransitions*.

Type *PageTransitions*(Enum), optional

property image: *libacbf.bookdata.BookData*
Gets the image data from the source.

Returns A *BookData* object.

Return type *BookData*

set_transition(*tr: Optional[str]*)
Set transition by string.

Parameters **tr** (*str | None*) – Transition value to be set. Pass None to remove.

add_textlayer(*lang: str*, **areas:* libacbf.body.TextArea) → *libacbf.body.TextLayer*
Add a text layer to the page.

Parameters

- **lang** (*str*) – The language of the text layer.
- ***areas** (*TextArea, optional*) – TextArea objects to fill the layer with.

Returns The newly created text layer.

Return type *TextLayer*

insert_frame(*index: int*, *points: List[Tuple[int, int]]*) → *libacbf.body.Frame*
Insert a frame at the index.

Parameters

- **index** (*int*) – Index to insert at.
- **points** (*List[Tuple[int, int]]*) – The points defining the frame.

Returns The newly created frame.

Return type *Frame*

append_frame(*points: List[Tuple[int, int]]*) → *libacbf.body.Frame*
Append a frame to the page.

Returns The newly created frame.

Return type *Frame*

add_jump(*target: int*, *points: List[Tuple[int, int]]*) → *libacbf.body.Jump*
Add a jump to the page.

Parameters

- **target** (*int*) – The target page. 0 is the cover page, 1 is the first page, 2 is the second page etc.
- **points** (*List[Tuple[int, int]]*) – The points defining the jump.

Returns The newly created jump.

Return type *Jump*

3.3.2 TextLayer

class libacbf.body.**TextLayer**(**areas:* libacbf.body.TextArea)
Bases: object

Defines a text layer drawn on a page.

See also:

Text Layer specifications.

text_areas
A list of *TextArea* objects in order (order matters for text-to-speech).

Type List[*TextArea*]

bgcolor
Defines the background colour of the text areas or inherits from *Page.bgcolor* if None.

Type str, optional

insert_textarea(*index: int*, *text: str*, *points: List[Tuple[int, int]]*) → *libacbf.body.TextArea*
Insert a text area at the index.

Parameters

- **index** (*int*) – Index to insert at.
- **text** (*str*) – Multiline text of the text area.
- **points** (*List[Tuple[int, int]]*) – The points that define the text area.

Returns

Return type The newly created text area.

append_textarea(*text: str*, *points: List[Tuple[int, int]]*) → *libacbf.body.TextArea*
Append a text area to the layer.

Parameters

- **text** (*str*) – Multiline text of the text area.
- **points** (*List[Tuple[int, int]]*) – The points that define the text area.

Returns

Return type The newly created text area.

3.3.3 TextArea

class libacbf.body.**TextArea**(*text: str*, *points: List[Tuple[int, int]]*)
Bases: object

Defines an area where text is drawn.

See also:

Text Area specifications.

points
A list of tuples as coordinates.

Type List[Tuple[int, int]]

text
A multiline string of what text to show in the are. Can have special tags for formatting.

... Bold letters.

<emphasis>...</emphasis> Italicised or cursive text.

<strikethrough>...</strikethrough> Striked-through text.

_{...} Subscript text.

^{...} Superscript text.

... A link. Internal or external.

Type str

bgcolor
Defines the background colour of the text area or inherits from *TextLayer.bgcolor* if None.

Type str, optional

rotation
Defines the rotation of the text layer.

Can be an integer from 0 to 360.

Type int, optional

type
The type of text area. Rendering can be changed based on type. Allowed values are defined in *TextAreas*.

Type *TextAreas*(Enum), optional

inverted
Whether text is rendered with inverted colour.

Type bool, optional

transparent
Whether text is drawn.

Type bool, optional

set_type(*ty: Optional[str]*)
Set type by string.

Parameters **ty** (*str | None*) – Type to set or None to remove.

3.3.4 Frame

class libacbf.body.**Frame**(*points: List[Tuple[int, int]]*)
Bases: object

A subsection of a page.

See also:

Frame specifications.

points
A list of tuples as coordinates.

Type List[Tuple[int, int]]

bgcolor
Defines the background colour for the page. Inherits from *Page.bgcolor* if None.

Type str, optional

3.3.5 Jump

class libacbf.body.**Jump**(*target: int*, *points: List[Tuple[int, int]]*, *book: ACBFBook*)
Bases: object

Clickable area on a page which navigates to another page.

See also:

body Info Jump specifications.

target
The target page index. Cover page is 0, first page is 1, second page is 2 and so on.

Type int

points
A list of tuples as coordinates.

Type List[Tuple[int, int]]

property **page**: *libacbf.body.Page*
Target page to go to when clicked.

3.4 Book Data

class libacbf.bookdata.**BookData**(*id: str*, *file_type: str*, *data: Union[str, bytes]*)
Bases: object

Binary data referenced or stored in the book.

See also:

Binary data specifications.

id
Name of the file with extension.

Type str

file_type
Mime type of the file.

Type str

data
The actual file's data.

Type bytes

3.5 Constants

Warning: The values of the enum members don't matter and there is no guarantee that they will never change. If you have to use it, use strings instead (case sensitive).

```
activity = AuthorActivities.Artist.name

# Check if value exists
_ = AuthorActivities["Writer"]  # No `KeyError` exception
_ = AuthorActivities["asdfgh"]  # `KeyError` exception is raised
```

3.5.1 AuthorActivities(Enum)

class libacbf.constants.**AuthorActivities**(*value*)

Bases: enum.Enum

List of accepted values for *Author.activity*.

Writer = 0

Adapter = 1

Artist = 2

Penciller = 3

Inker = 4

Colorist = 5

Letterer = 6

CoverArtist = 7

Photographer = 8

Editor = 9

AssistantEditor = 10

Translator = 11

Other = 12

3.5.2 Genres(Enum)

class libacbf.constants.**Genres**(*value*)

Bases: enum.Enum

List of accepted values for keys of *book_info.genres*.

adult = 0

adventure = 1

alternative = 2

biography = 3

caricature = 4

children = 5

computer = 6

crime = 7

education = 8

fantasy = 9

history = 10

horror = 11

humor = 12

manga = 13

military = 14

mystery = 15

non_fiction = 16

politics = 17

real_life = 18

religion = 19

romance = 20

science_fiction = 21

sports = 22

superhero = 23

western = 24

other = 25

3.5.3 TextAreas(Enum)

class libacbf.constants.**TextAreas**(*value*)

Bases: enum.Enum

Types of text areas. Used by *TextArea.type*.

speech = 0

commentary = 1

formal = 2

letter = 3

code = 4

heading = 5

audio = 6

thought = 7

sign = 8

3.5.4 PageTransitions(Enum)

class libacbf.constants.**PageTransitions**(*value*)
Bases: enum.Enum

Allowed values for *Page.transition*.

fade = 0

blend = 1

scroll_right = 2

scroll_down = 3

none = 4

3.5.5 ImageRefType(Enum)

class libacbf.constants.**ImageRefType**(*value*)
Bases: enum.Enum

Types of image references. Used by *Page.ref_type*.

Embedded = 0

SelfArchived = 1

Archived = 2

Local = 3

URL = 4

3.5.6 ArchiveTypes(Enum)

class libacbf.constants.**ArchiveTypes**(*value*)
Bases: enum.Enum

The type of the source archive file. Used by *ArchiveReader.type*.

Zip = 0

SevenZip = 1

Tar = 2

Rar = 3

3.6 Exceptions

exception libacbf.exceptions.**UnsupportedArchive**(*message: str = 'File is not a supported archive type.'*, **args*)
Bases: Exception

exception libacbf.exceptions.**InvalidBook**(*message: str = 'File is not an ACBF Ebook.'*, **args*)
Bases: Exception

exception libacbf.exceptions.**EditRARArchiveError**(*message: str = 'Editing RAR Archives is not supported.'*, **args*)

Bases: Exception

3.7 libacbf.archivereader module

libacbf.archivereader.**get_archive_type**(*file: Union[str, pathlib.Path, BinaryIO]*) → *libacbf.constants.ArchiveTypes*

Get the type of archive.

Parameters **file** (*str | pathlib.Path | BinaryIO*) – File to check.

Returns Returns *ArchiveTypes* enum.

Return type *ArchiveTypes*(Enum)

Raises ***UnsupportedArchive*** – Raised if file is not of a supported archive type.

class libacbf.archivereader.**ArchiveReader**(*file: Union[str, pathlib.Path, BinaryIO]*, *mode: Literal['r', 'w'] = 'r'*)

Bases: object

This can read and write Zip, 7Zip and Tar archives. Rar archives are read-only.

Notes

Writing and creating archives uses the default options for each type. You cannot use this module to change compression levels or other options.

Parameters

- **file** (*str | pathlib.Path | BinaryIO*) – Archive file to be used.
- **mode** (*'r' | 'w'*) – Mode to open file in. Can be 'r' for read-only or 'w' for read-write. Nothing is overwritten.

archive
The archive being used.

Type zipfile.ZipFile | tarfile.TarFile | py7zr.SevenZipFile | rarfile.RarFile

type
The type of archive. See enum for possible types.

Type *ArchiveTypes*

mode
Mode to open file in. Can be 'r' for read-only or 'w' for read-write. Nothing is overwritten.

Type 'r' | 'w'

_extract
The contents of the archive are extracted to a temporary directory in write mode only and this is used for listing, reading and writing. It is created in the same directory as the archive or, if the path is not found, it is created in the system temp directory.

Type tempfile.TemporaryDirectory | None

_arc_path
The path to the temporary directory the archive is extracted to in write mode.

Type pathlib.Path | None

_source
The file passed in.

Type str | Path | BinaryIO

__init__(*file: Union[str, pathlib.Path, BinaryIO]*, *mode: Literal['r', 'w'] = 'r'*)

property filepath: Optional[pathlib.Path]
Path to the archive file. Returns `None` if it does not have a path.

property filename: Optional[str]
Name of the archive file. Returns `None` if it does not have a path.

_get_acbf_file() → Optional[str]
Returns the name of the first file with the `.acbf` extension at the root level of the archive or `None` if no file is found.

list_files() → Set[str]
Returns a list of all the names of the files in the archive.

list_dirs() → Set[str]
Returns a list of all the directories in the archive.

read(*target: str*) → Optional[bytes]
Get file as bytes from archive.

Parameters **target** (*str*) – Path relative to root of archive.

Returns Contents of file.

Return type bytes

write(*target: Union[str, pathlib.Path, bytes]*, *arcname: Optional[str] = None*)
Write file to archive.

Parameters

- **target** (*str | Path | bytes*) – File to be written. Reads a file on disk if string or path is passed. Writes data directly if bytes is passed.
- **arcname** (*str, default=Name of target file*) – Name of file in archive.

delete(*target: Union[str, pathlib.Path]*, *recursive: bool = False*)
File to delete from archive.

Parameters

- **target** (*str | Path*) – Path of file to delete relative to root of archive.
- **recursive** (*bool, default=False*) – Whether to remove directories recursively.

__module__ = 'libacbf.archivereader'

__weakref__
list of weak references to the object (if defined)

close()
Close archive file. Save changes if writeable.

__enter__()

__exit__(*exception_type*, *exception_value*, *traceback*)

CHAPTER

FOUR

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

INDEX

Symbols

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

R

S

T

U

V

W

Z