libkiwix

libzim-team

Dec 01, 2021

CONTENTS:

CHAPTER

ONE

LIBKIWIX PROGRAMMING

1.1 Introduction

libkiwix is written in C++. To use the library, you need the include files of libkiwix have to link against libzim.

Errors are handled with exceptions. When something goes wrong, libzim throws an error, which is always derived from std::exception.

All classes are defined in the namespace kiwix.

libkiwix is a set of tools to manage zim files and provide some common functionnality. While libkiwix has some wrappers around libzim classes, they are deprecated and will be removed in the future.

CHAPTER

TWO

REFERENCE API

2.1 Class Hierarchy

2.2 File Hierarchy

2.3 Full API

2.3.1 Namespaces

Namespace kiwix

Contents

Classes

- *Class HumanReadableNameMapper*
- *Class IdNameMapper*
- *Class KiwixServe*
- *Class Library*
- *Class LibraryBase*
- *Class LibraryManipulator*
- *Class LibXMLDumper*
- *Class Manager*
- *Class NameMapper*
- *Class NoEntry*
- *Class OPDSDumper*
- *Class Reader*
- *Class Result*
- *Class Searcher*
- *Class SearchRenderer*
- *Class Server*
- *Class SuggestionItem*
- *Class UpdatableNameMapper*

Enums

- *Enum supportedListMode*
- *Enum supportedListSortBy*

Functions

- *Function kiwix::appendToDirectory*
- *Function kiwix::computeAbsolutePath*
- *Function kiwix::computeRelativePath*
- *Function kiwix::converta2toa3*
- *Function kiwix::fileExists*
- *Function kiwix::getCurrentDirectory*
- *Function kiwix::getDataDirectory*
- *Function kiwix::getExecutablePath*
- *Function kiwix::getFileContent*
- *Function kiwix::getLastPathElement*
- *Function kiwix::getMimeTypeForFile*

- *Function kiwix::getSearchUrl*
- *Function kiwix::isRelativePath*
- *Function kiwix::removeLastPathElement*
- *Function kiwix::sleep*
- *Function kiwix::split*

Typedefs

- *Typedef kiwix::offset_type*
- *Typedef kiwix::size_type*
- *Typedef kiwix::SuggestionsList_t*

Namespace pugi

Namespace std

Namespace zim

2.3.2 Classes and Structs

Struct DownloadedFile

- Defined in file_include_downloader.h

Struct Documentation

struct kiwix::**DownloadedFile**

Public Functions

inline **DownloadedFile**()

Public Members

bool **success**

std::string **path**

Struct LibraryBase::Entry

- Defined in file_include_library.h

Nested Relationships

This struct is a nested type of *Class LibraryBase*.

Inheritance Relationships

Base Type

- `public kiwix::Book` (*Class Book*)

Struct Documentation

struct kiwix::*LibraryBase*::**Entry** : **public** kiwix::*Book*

Public Members

LibraryRevision **lastUpdatedRevision** = 0

Class AriaError

- Defined in file_include_downloader.h

Inheritance Relationships

Base Type

- `public runtime_error`

Class Documentation

class kiwix::**AriaError** : **public** runtime_error

Public Functions

inline **AriaError**(**const** std::string &*message*)

Class Book

- Defined in file_include_book.h

Nested Relationships

Nested Types

- *Class Book::Illustration*

Inheritance Relationships

Derived Type

- `public kiwix::LibraryBase::Entry` (*Struct LibraryBase::Entry*)

Class Documentation

class `kiwix::Book`

A class to store information about a book (a zim file)

Subclassed by *kiwix::LibraryBase::Entry*

Public Types

typedef std::vector<std::shared_ptr<**const** *Illustration*>> **Illustrations**

Public Functions

Book()

~Book()

bool **update**(**const** *Book* &*other*)

void **update**(**const** *Reader* &*reader*)

void **update**(**const** zim::Archive &*archive*)

void **updateFromXml**(**const** pugi::xml_node &*node*, **const** std::string &*baseDir*)

void **updateFromOpds**(**const** pugi::xml_node &*node*, **const** std::string &*urlHost*)

std::string **getHumanReadableIdFromPath**() **const**

inline bool **readOnly**() **const**

inline const std::string &**getId**() **const**

inline const std::string &**getPath**() **const**

inline bool **isPathValid**() **const**

inline const std::string &**getTitle**() **const**

inline const std::string &**getDescription**() **const**

inline const std::string &**getLanguage**() **const**

inline const std::string &**getCreator**() **const**

inline const std::string &**getPublisher**() **const**

inline const std::string &**getDate**() **const**

inline const std::string &**getUrl**() **const**

inline const std::string &**getName**() **const**

std::string **getCategory**() **const**

inline const std::string &**getTags**() **const**

std::string **getTagStr**(**const** std::string &*tagName*) **const**

bool **getTagBool**(**const** std::string &*tagName*) **const**

inline const std::string &**getFlavour**() **const**

inline const std::string &**getOrigId**() **const**

inline const uint64_t &**getArticleCount**() **const**

inline const uint64_t &**getMediaCount**() **const**

inline const uint64_t &**getSize**() **const**

const std::string &**getFavicon**() **const**

const std::string &**getFaviconUrl**() **const**

const std::string &**getFaviconMimeType**() **const**

Illustrations **getIllustrations**() **const**

inline const std::string &**getDownloadId**() **const**

inline void **setReadOnly**(bool *readOnly*)

inline void **setId**(**const** std::string &*id*)

void **setPath**(**const** std::string &*path*)

inline void **setPathValid**(bool *valid*)

inline void **setTitle**(**const** std::string &*title*)

inline void **setDescription**(**const** std::string &*description*)

inline void **setLanguage**(**const** std::string &*language*)

inline void **setCreator**(**const** std::string &*creator*)

inline void **setPublisher**(**const** std::string &*publisher*)

inline void **setDate**(**const** std::string &*date*)

inline void **setUrl**(**const** std::string &*url*)

inline void **setName**(**const** std::string &*name*)

inline void **setFlavour**(**const** std::string &*flavour*)

inline void **setTags**(**const** std::string &*tags*)

inline void **setOrigId**(**const** std::string &*origId*)

inline void **setArticleCount**(uint64_t *articleCount*)

inline void **setMediaCount** (uint64_t *mediaCount*)

inline void **setSize** (uint64_t *size*)

inline void **setDownloadId** (**const** std::string &*downloadId*)

Protected Attributes

std::string **m_id**

std::string **m_downloadId**

std::string **m_path**

bool **m_pathValid** = false

std::string **m_title**

std::string **m_description**

std::string **m_category**

std::string **m_language**

std::string **m_creator**

std::string **m_publisher**

std::string **m_date**

std::string **m_url**

std::string **m_name**

std::string **m_flavour**

std::string **m_tags**

std::string **m_origId**

uint64_t **m_articleCount** = 0

uint64_t **m_mediaCount** = 0

bool **m_readOnly** = false

uint64_t **m_size** = 0

Illustrations **m_illustrations**

Protected Static Attributes

static const *Illustration* **missingDefaultIllustration**

class Illustration

Public Functions

const std::string &**getData**() **const**

Public Members

uint16_t **width** = 48

uint16_t **height** = 48

std::string **mimeType**

std::string **url**

Class Book::Illustration

- Defined in file_include_book.h

Nested Relationships

This class is a nested type of *Class Book*.

Class Documentation

class kiwix::*Book*::**Illustration**

Public Functions

const std::string &**getData**() **const**

Public Members

uint16_t **width** = 48

uint16_t **height** = 48

std::string **mimeType**

std::string **url**

Class Bookmark

- Defined in file_include_bookmark.h

Class Documentation

class kiwix::**Bookmark**

A class to store information about a bookmark (an article in a book)

Public Functions

Bookmark()

~Bookmark()

void **updateFromXml**(**const** pugi::xml_node &*node*)

inline const std::string &**getBookId**() **const**

inline const std::string &**getBookTitle**() **const**

inline const std::string &**getUrl**() **const**

inline const std::string &**getTitle**() **const**

inline const std::string &**getLanguage**() **const**

inline const std::string &**getDate**() **const**

inline void **setBookId**(**const** std::string &*bookId*)

inline void **setBookTitle**(**const** std::string &*bookTitle*)

inline void **setUrl**(**const** std::string &*url*)

inline void **setTitle**(**const** std::string &*title*)

inline void **setLanguage**(**const** std::string &*language*)

inline void **setDate**(**const** std::string &*date*)

Protected Attributes

std::string **m_bookId**

std::string **m_bookTitle**

std::string **m_url**

std::string **m_title**

std::string **m_language**

std::string **m_date**

Class Download

- Defined in file_include_downloader.h

Class Documentation

class kiwix::**Download**

Public Types

enum StatusResult

Values:

enumerator K_ACTIVE

enumerator K_WAITING

enumerator K_PAUSED

enumerator K_ERROR

enumerator K_COMPLETE

enumerator K_REMOVED

enumerator K_UNKNOWN

Public Functions

inline Download()

inline Download(std::shared_ptr<Aria2> *p_aria*, std::string *did*)

void **updateStatus**(bool *follow* = false)

void **pauseDownload**()

void **resumeDownload**()

void **cancelDownload**()

inline *StatusResult* **getStatus**()

inline std::string **getDid**()

inline std::string **getFollowedBy**()

inline uint64_t **getTotalLength**()

inline uint64_t **getCompletedLength**()

inline uint64_t **getDownloadSpeed**()

inline uint64_t **getVerifiedLength**()

inline std::string **getPath**()

inline std::vector<std::string> &**getUris**()

Protected Attributes

std::shared_ptr<Aria2> **mp_aria**

StatusResult **m_status**

std::string **m_did** = ""

std::string **m_followedBy** = ""

uint64_t **m_totalLength**

uint64_t **m_completedLength**

uint64_t **m_downloadSpeed**

uint64_t **m_verifiedLength**

std::vector<std::string> **m_uris**

std::string **m_path**

Class Downloader

- Defined in file_include_downloader.h

Class Documentation

class kiwix::**Downloader**
A tool to download things.

Public Functions

Downloader()

virtual ~Downloader()

void **close**()

Download ***startDownload**(**const** std::string *&uri*, **const** std::vector<std::pair<std::string, std::string>> *&options* = {})

Download ***getDownload**(**const** std::string *&did*)

inline size_t **getNbDownload**()

std::vector<std::string> **getDownloadIds**()

Class Entry

- Defined in file_include_entry.h

Class Documentation

class kiwix::**Entry**

A entry represent an.. entry in a zim file.

Public Functions

Entry(zim::Entry *entry*)

Construct an entry making reference to an zim article.

Parameters **article** – a zim::Article object

virtual **~Entry**() = default

inline std::string **getPath**() **const**

Get the path of the entry.

The path is the "key" of an entry.

Returns the path of the entry.

inline std::string **getTitle**() **const**

Get the title of the entry.

Returns the title of the entry.

inline std::string **getContent**() **const**

Get the content of the entry.

The string is a copy of the content. If you don't want to do a copy, use get_blob.

Returns the content of the entry.

inline zim::Blob **getBlob**(*offset_type offset* = 0) **const**

Get the blob of the entry.

A blob make reference to the content without copying it.

Parameters **offset** – The starting offset of the blob.

Returns the blob of the entry.

inline zim::Blob **getBlob**(*offset_type offset*, *size_type size*) **const**

Get the blob of the entry.

A blob make reference to the content without copying it.

Parameters

- **offset** – The starting offset of the blob.
- **size** – The size of the blob.

Returns the blob of the entry.

inline zim::Item::DirectAccessInfo **getDirectAccessInfo**() **const**

Get the info for direct access to the content of the entry.

Some entry (ie binary ones) have their content plain stored in the zim file. Knowing the offset where the content is stored an user can directly read the content in the zim file bypassing the libkiwix/libzim.

Returns A pair specifying where to read the content. The string is the real file to read (may be different that .zim file if zim is cut). The offset is the offset to read in the file. Return <"",0> if is not possible to read directly.

size_type **getSize**() **const**
Get the size of the entry.

Returns the size of the entry.

std::string **getMimetype**() **const**
Get the mime_type of the entry.

Returns the mime_type of the entry.

bool **isRedirect**() **const**
Get if the entry is a redirect entry.

Returns True if the entry is a redirect.

bool **isLinkTarget**() **const**
Get if the entry is a link target entry.

Returns True if the entry is a link target.

bool **isDeleted**() **const**
Get if the entry is a deleted entry.

Returns True if the entry is a deleted entry.

Entry **getRedirectEntry**() **const**
Get the entry pointed by this entry.

Throws ***NoEntry*** – if the entry is not a redirected entry.

Returns the entry pointed.

Entry **getFinalEntry**() **const**
Get the final entry pointed by this entry.

Follow the redirection until a "not redirecting" entry is found. If the entry is not a redirected entry, return the entry itself.

Returns the final entry.

inline **const** zim::Entry &**getZimEntry**() **const**
Get the zim entry wrapped by this (kiwix) entry

Returns the zim entry

Class Filter

- Defined in file_include_library.h

Class Documentation

class kiwix::**Filter**

Public Types

using Tags = std::vector<std::string>

Public Functions

Filter()

~Filter() = default

Filter &**local**(bool *accept*)
Set the filter to check local.

A local book is a book with a path. If accept is true, only local book are accepted. If accept is false, only non local book are accepted.

Filter &**remote**(bool *accept*)
Set the filter to check remote.

A remote book is a book with a url. If accept is true, only remote book are accepted. If accept is false, only non remote book are accepted.

Filter &**valid**(bool *accept*)
Set the filter to check validity.

A valid book is a book with a path pointing to a existing zim file. If accept is true, only valid book are accepted. If accept is false, only non valid book are accepted.

Filter &**acceptTags**(**const** *Tags* &*tags*)
Set the filter to only accept book with corresponding tag.

Filter &**rejectTags**(**const** *Tags* &*tags*)

Filter &**category**(std::string *category*)

Filter &**lang**(std::string *lang*)

Filter &**publisher**(std::string *publisher*)

Filter &**creator**(std::string *creator*)

Filter &**maxSize**(size_t *size*)

Filter &**query**(std::string *query*, bool *partial* = true)

Filter &**name**(std::string *name*)

bool **hasQuery**() **const**

inline **const** std::string &**getQuery**() **const**

inline bool **queryIsPartial**() **const**

bool **hasName**() **const**

inline **const** std::string &**getName**() **const**

bool **hasCategory**() **const**

inline **const** std::string &**getCategory**() **const**

bool **hasLang**() **const**

inline **const** std::string &**getLang**() **const**

bool **hasPublisher**() **const**

inline const std::string &**getPublisher**() **const**

bool **hasCreator**() **const**

inline const std::string &**getCreator**() **const**

inline const *Tags* &**getAcceptTags**() **const**

inline const *Tags* &**getRejectTags**() **const**

Class HumanReadableNameMapper

- Defined in file_include_name_mapper.h

Inheritance Relationships

Base Type

- `public kiwix::NameMapper` (*Class NameMapper*)

Class Documentation

class `kiwix::`**HumanReadableNameMapper** : **public** kiwix::*NameMapper*

Public Functions

HumanReadableNameMapper(kiwix::*Library* &*library*, bool *withAlias*)

virtual ~HumanReadableNameMapper() = default

virtual std::string **getNameForId**(**const** std::string &*id*) **const**

virtual std::string **getIdForName**(**const** std::string &*name*) **const**

Class IdNameMapper

- Defined in file_include_name_mapper.h

Inheritance Relationships

Base Type

- `public kiwix::NameMapper` (*Class NameMapper*)

Class Documentation

class kiwix::**IdNameMapper** : **public** kiwix::*NameMapper*

Public Functions

inline virtual std::string **getNameForId**(**const** std::string &*id*) **const**

inline virtual std::string **getIdForName**(**const** std::string &*name*) **const**

Class KiwixServe

- Defined in file_include_kiwixserve.h

Class Documentation

class kiwix::**KiwixServe**

Public Functions

KiwixServe(**const** std::string &*libraryPath*, int *port* = 8181)

~KiwixServe()

void **run**()

void **shutDown**()

bool **isRunning**()

inline int **getPort**()

int **setPort**(int *port*)

Class Library

- Defined in file_include_library.h

Inheritance Relationships

Base Type

- private kiwix::LibraryBase (*Class LibraryBase*)

Class Documentation

class kiwix::**Library** : **private** kiwix::*LibraryBase*

A *Library* store several books.

Public Types

typedef LibraryRevision **Revision**

typedef std::vector<std::string> **BookIdCollection**

typedef std::map<std::string, int> **AttributeCounts**

Public Functions

Library()

~Library()

Library(**const** *Library*&) = delete

Library is not a copiable object. However it can be moved.

Library(*Library*&&)

void **operator=**(**const** *Library*&) = delete

Library &**operator=**(*Library*&&)

bool **addBook**(**const** *Book* &*book*)

Add a book to the library.

If a book already exist in the library with the same id, update the existing book instead of adding a new one.

Parameters **book** – The book to add.

Returns True if the book has been added. False if a book has been updated.

inline bool **addOrUpdateBook**(**const** *Book* &*book*)

A self-explanatory alias for *addBook()*

void **addBookmark**(**const** *Bookmark* &*bookmark*)

Add a bookmark to the library.

Parameters **bookmark** – the book to add.

bool **removeBookmark**(**const** std::string &*zimId*, **const** std::string &*url*)

Remove a bookmarkk

Parameters

- **zimId** – The zimId of the bookmark.
- **url** – The url of the bookmark.

Returns True if the bookmark has been removed.

const *Book* &**getBookById**(**const** std::string &*id*) **const**

const *Book* &**getBookByPath**(**const** std::string &*path*) **const**

Book **getBookByIdThreadSafe**(**const** std::string &*id*) **const**

std::shared_ptr<*Reader*> **getReaderById**(**const** std::string &*id*)

std::shared_ptr<zim::Archive> **getArchiveById**(**const** std::string &*id*)

bool **removeBookById**(**const** std::string &*id*)
Remove a book from the library.

Parameters **id** – the id of the book to remove.

Returns True if the book were in the lirbrary and has been removed.

bool **writeToFile**(**const** std::string &*path*) **const**
Write the library to a file.

Parameters **path** – the path of the file to write to.

Returns True if the library has been correctly saved.

bool **writeBookmarksToFile**(**const** std::string &*path*) **const**
Write the library bookmarks to a file.

Parameters **path** – the path of the file to write to.

Returns True if the library has been correctly saved.

unsigned int **getBookCount**(**const** bool *localBooks*, **const** bool *remoteBooks*) **const**
Get the number of book in the library.

Parameters

- **localBooks** – If we must count local books (books with a path).
- **remoteBooks** – If we must count remote books (books with an url)

Returns The number of books.

std::vector<std::string> **getBooksLanguages**() **const**
Get all languagues of the books in the library.

Returns A list of languages.

AttributeCounts **getBooksLanguagesWithCounts**() **const**
Get all languagues of the books in the library with counts.

Returns A list of languages with the count of books in each language.

std::vector<std::string> **getBooksCategories**() **const**
Get all categories of the books in the library.

Returns A list of categories.

std::vector<std::string> **getBooksCreators**() **const**
Get all book creators of the books in the library.

Returns A list of book creators.

std::vector<std::string> **getBooksPublishers**() **const**
Get all book publishers of the books in the library.

Returns A list of book publishers.

const std::vector<kiwix::*Bookmark*> **getBookmarks**(bool *onlyValidBookmarks* = true) **const**
Get all bookmarks.

Returns A list of bookmarks

BookIdCollection **getBooksIds**() **const**
Get all book ids of the books in the library.

Returns A list of book ids.

BookIdCollection **filter**(**const** std::string &*search*) **const**
Filter the library and generate a new one with the keep elements.

This is equivalent to `listBookIds(ALL, UNSORTED, search)`.

Parameters **search** – List only books with search in the title or description.

Returns The list of bookIds corresponding to the query.

BookIdCollection **filter**(**const** *Filter* &*filter*) **const**
Filter the library and return the id of the keep elements.

Parameters **filter** – The filter to use.

Returns The list of bookIds corresponding to the filter.

void **sort**(*BookIdCollection* &*bookIds*, *supportedListSortBy* *sortBy*, bool *ascending*) **const**
Sort (in place) bookIds using the given comparator.

Parameters

- **bookIds** – the list of book Ids to sort
- **comparator** – how to sort the books

Returns The sorted list of books

BookIdCollection **listBooksIds**(int *supportedListMode* = ALL, *supportedListSortBy* *sortBy* = UNSORTED, **const** std::string &*search* = "", **const** std::string &*language* = "", **const** std::string &*creator* = "", **const** std::string &*publisher* = "", **const** std::vector<std::string> &*tags* = {}, size_t *maxSize* = 0) **const**
List books in the library.

Parameters

- **mode** – The mode of listing :
  - LOCAL : list only local books (with a path).
  - REMOTE : list only remote books (with an url).
  - VALID : list only valid books (without a path or with a path pointing to a valid zim file).
  - NOLOCAL : list only books without valid path.
  - NOREMOTE : list only books without url.
  - NOVALID : list only books not valid.
  - ALL : Do not do any filter (LOCAL or REMOTE)
  - Flags can be combined.
- **sortBy** – Attribute to sort by the book list.
- **search** – List only books with search in the title, description.
- **language** – List only books in this language.
- **creator** – List only books of this creator.
- **publisher** – List only books of this publisher.
- **maxSize** – Do not list book bigger than maxSize. Set to 0 to cancel this filter.

Returns The list of bookIds corresponding to the query.

LibraryRevision **getRevision**() **const**
Return the current revision of the library.

The revision of the library is updated (incremented by one) only by the *addBook()* operation.

Returns Current revision of the library.

uint32_t **removeBooksNotUpdatedSince**(LibraryRevision *rev*)
Remove books that have not been updated since the specified revision.

Parameters **rev** – the library revision to use

Returns Count of books that were removed by this operation.

Friends

friend class OPDSDumper

friend class libXMLDumper

Class LibraryBase

- Defined in file_include_library.h

Nested Relationships

Nested Types

- *Struct LibraryBase::Entry*

Inheritance Relationships

Derived Type

- `private kiwix::Library` (*Class Library*)

Class Documentation

class `kiwix::`**LibraryBase**
This class is not part of the libkiwix API. Its only purpose is to simplify the implementation of the *Library*'s move operations and avoid bugs should new data members be added to *Library*.

Subclassed by *kiwix::Library*

Protected Types

typedef uint64_t **LibraryRevision**

Protected Functions

LibraryBase()

~LibraryBase()

LibraryBase(*LibraryBase*&&)

LibraryBase &**operator=**(*LibraryBase*&&)

Protected Attributes

LibraryRevision **m_revision**

std::map<std::string, *Entry*> **m_books**

std::map<std::string, std::shared_ptr<*Reader*>> **m_readers**

std::map<std::string, std::shared_ptr<zim::Archive>> **m_archives**

std::vector<kiwix::*Bookmark*> **m_bookmarks**

std::unique_ptr<BookDB> **m_bookDB**

struct Entry : **public** kiwix::*Book*

Public Members

LibraryRevision **lastUpdatedRevision** = 0

Class LibraryManipulator

- Defined in file_include_manager.h

Class Documentation

class kiwix::**LibraryManipulator**

Public Functions

explicit LibraryManipulator(*Library* **library*)

virtual ~LibraryManipulator()

inline *Library* &**getLibrary**() **const**

bool **addBookToLibrary**(**const** *Book* &*book*)

void **addBookmarkToLibrary**(**const** *Bookmark* &*bookmark*)

uint32_t **removeBooksNotUpdatedSince**(*Library*::*Revision rev*)

Protected Functions

virtual void **bookWasAddedToLibrary**(**const** *Book* &*book*)

virtual void **bookmarkWasAddedToLibrary**(**const** *Bookmark* &*bookmark*)

virtual void **booksWereRemovedFromLibrary**()

Class LibXMLDumper

- Defined in file_include_libxml_dumper.h

Class Documentation

class kiwix::**LibXMLDumper**

A tool to dump a *Library* into a basic library.xml

Public Functions

LibXMLDumper() = default

LibXMLDumper(**const** *Library* **library*)

~LibXMLDumper()

std::string **dumpLibXMLContent**(**const** std::vector<std::string> &*bookIds*)

Dump the library.xml

Parameters **id** – The id of the library.

Returns The library.xml content.

std::string **dumpLibXMLBookmark**()

Dump the bookmark of the library.

Returns The bookmark.xml content.

inline void **setBaseDir**(**const** std::string &*baseDir*)

Set the base directory used.

Parameters **baseDir** – the base directory to use.

inline void **setLibrary**(**const** *Library* **library*)

Set the library to dump.

Parameters **library** – The library to dump.

Protected Attributes

const kiwix::*Library* ***library**

std::string **baseDir**

Class Manager

- Defined in file_include_manager.h

Class Documentation

class kiwix::**Manager**

A tool to manage a *Library*.

Public Types

typedef std::vector<std::string> **Paths**

Public Functions

explicit Manager(*LibraryManipulator* **manipulator*)

explicit Manager(*Library* **library*)

bool **readFile**(**const** std::string &*path*, bool *readOnly* = true, bool *trustLibrary* = true)

Read a `library.xml` and add book in the file to the library.

Parameters

- **path** – The (utf8) path to the `library.xml`.
- **readOnly** – Set if the libray path could be overwritten latter with updated content.
- **trustLibrary** – use book metadata coming from XML.

Returns True if file has been properly parsed.

void **reload**(**const** *Paths* &*paths*)

Sync the contents of the library with one or more `library.xml` files.

The metadata of the library files is trusted unconditionally. Any books not present in the input library.xml files are removed from the library.

Parameters **paths** – The (utf8) paths to the `library.xml` files.

bool **readXml**(**const** std::string &*xml*, **const** bool *readOnly* = true, **const** std::string &*libraryPath* = "", bool *trustLibrary* = true)

Load a library content store in the string.

Parameters

- **xml** – The content corresponding of the library xml
- **readOnly** – Set if the libray path could be overwritten latter with updated content.
- **libraryPath** – The library path (used to resolve relative path)

Returns True if the content has been properly parsed.

bool **readOpds**(**const** std::string &*content*, **const** std::string &*urlHost*)

Load a library content stored in a OPDS stream.

Parameters

- **content** – The content of the OPDS stream.
- **readOnly** – Set if the library path could be overwritten later with updated content.

- **libraryPath** – The library path (used to resolve relative path)

Returns True if the content has been properly parsed.

bool **readBookmarkFile**(**const** std::string &*path*)

Load a bookmark file.

Parameters **path** – The path of the file to read.

Returns True if the content has been properly parsed.

std::string **addBookFromPathAndGetId**(**const** std::string &*pathToOpen*, **const** std::string &*pathToSave* = "", **const** std::string &*url* = "", **const** bool *checkMetaData* = false)

Add a book to the library.

Parameters

- **pathToOpen** – The path to the zim file to add.
- **pathToSave** – The path to store in the library in place of pathToOpen.
- **url** – The url of the book to store in the library.
- **checMetaData** – Tell if we check metadata before adding book to the library.

Returns The id of the book if the book has been added to the library. Else, an empty string.

bool **addBookFromPath**(**const** std::string &*pathToOpen*, **const** std::string &*pathToSave* = "", **const** std::string &*url* = "", **const** bool *checkMetaData* = false)

Add a book to the library.

Parameters

- **pathToOpen** – The path to the zim file to add.
- **pathToSave** – The path to store in the library in place of pathToOpen.
- **url** – The url of the book to store in the library.
- **checMetaData** – Tell if we check metadata before adding book to the library.

Returns True if the book has been added to the library.

Public Members

std::string **writableLibraryPath**

bool **m_hasSearchResult** = false

uint64_t **m_totalBooks** = 0

uint64_t **m_startIndex** = 0

uint64_t **m_itemsPerPage** = 0

Protected Functions

bool **readBookFromPath**(**const** std::string &*path*, *Book* **book*)

bool **parseXmlDom**(**const** pugi::xml_document &*doc*, bool *readOnly*, **const** std::string &*libraryPath*, bool *trustLibrary*)

bool **parseOpdsDom**(**const** pugi::xml_document &*doc*, **const** std::string &*urlHost*)

Protected Attributes

std::shared_ptr<kiwix::*LibraryManipulator*> **manipulator**

Class NameMapper

- Defined in file_include_name_mapper.h

Inheritance Relationships

Derived Types

- `public kiwix::HumanReadableNameMapper` (*Class HumanReadableNameMapper*)
- `public kiwix::IdNameMapper` (*Class IdNameMapper*)
- `public kiwix::UpdatableNameMapper` (*Class UpdatableNameMapper*)

Class Documentation

class `kiwix::`**NameMapper**

Subclassed by *kiwix::HumanReadableNameMapper*, *kiwix::IdNameMapper*, *kiwix::UpdatableNameMapper*

Public Functions

virtual ~NameMapper() = default

virtual std::string **getNameForId**(**const** std::string &*id*) **const** = 0

virtual std::string **getIdForName**(**const** std::string &*name*) **const** = 0

Class NoEntry

- Defined in file_include_entry.h

Inheritance Relationships

Base Type

- `public exception`

Class Documentation

class NoEntry : **public** exception

Class OPDSDumper

- Defined in file_include_opds_dumper.h

Class Documentation

class kiwix::**OPDSDumper**

A tool to dump a *Library* into a opds stream.

Public Functions

OPDSDumper() = default

OPDSDumper(*Library* *library)

~OPDSDumper()

std::string **dumpOPDSFeed**(**const** std::vector<std::string> &*bookIds*, **const** std::string &*query*) **const**

Dump the OPDS feed.

Parameters

- **bookIds** – the ids of the books to include in the feed
- **query** – the query used to obtain the list of book ids

Returns The OPDS feed.

std::string **dumpOPDSFeedV2**(**const** std::vector<std::string> &*bookIds*, **const** std::string &*query*, bool *partial*) **const**

Dump the OPDS feed.

Parameters

- **bookIds** – the ids of the books to include in the feed
- **query** – the query used to obtain the list of book ids
- **partial** – whether the feed should include partial or complete entries

Returns The OPDS feed.

std::string **dumpOPDSCompleteEntry**(**const** std::string &*bookId*) **const**

Dump the OPDS complete entry document.

Parameters **bookId** – the id of the book

Returns The OPDS complete entry document.

std::string **categoriesOPDSFeed**() **const**
Dump the categories OPDS feed.

Returns The OPDS feed.

std::string **languagesOPDSFeed**() **const**
Dump the languages OPDS feed.

Returns The OPDS feed.

inline void **setLibraryId**(**const** std::string &*id*)
Set the id of the library.

Parameters **id** – the id to use.

inline void **setRootLocation**(**const** std::string &*rootLocation*)
Set the root location used when generating url.

Parameters **rootLocation** – the root location to use.

void **setOpenSearchInfo**(int *totalResult*, int *startIndex*, int *count*)
Set some informations about the search results.

Parameters

- **totalResult** – the total number of results of the search.
- **startIndex** – the start index of the result.
- **count** – the number of result of the current set (or page).

Protected Attributes

kiwix::*Library* ***library**

std::string **libraryId**

std::string **rootLocation**

int **m_totalResults**

int **m_startIndex**

int **m_count**

Class Reader

- Defined in file_include_reader.h

Class Documentation

class kiwix::**Reader**

Public Functions

explicit Reader(**const** string *zimFilePath*)
Create a *Reader* to read a zim file specified by zimFilePath.

Parameters **zimFilePath** – The path to the zim file to read. The zim file can be splitted (.zimaa, .zimab, ...). In this case, the file path must still point to the unsplitted path as if the file were not splitted (.zim extesion).

explicit Reader(**const** std::shared_ptr<zim::Archive> *archive*)
Create a *Reader* to read a zim file given by the Archive.

Parameters **archive** – The shared pointer to the Archive object.

explicit Reader(int *fd*)

Reader(int *fd*, zim::offset_type *offset*, zim::size_type *size*)

~Reader() = default

unsigned int **getArticleCount**() **const**
Get the number of "displayable" entries in the zim file.

Returns If the zim file has a /M/Counter metadata, return the number of entries with the 'text/html' MIMEtype specified in the metadata. Else return the number of entries in the 'A' namespace.

unsigned int **getMediaCount**() **const**
Get the number of media in the zim file.

Returns If the zim file has a /M/Counter metadata, return the number of entries with the 'image/jpeg', 'image/gif' and 'image/png' in the metadata. Else return the number of entries in the 'I' namespace.

unsigned int **getGlobalCount**() **const**
Get the number of all entries in the zim file.

Returns Return the number of all the entries, whatever their MIMEtype or their namespace.

string **getZimFilePath**() **const**
Get the path of the zim file.

Returns the path of the zim file as given in the constructor.

string **getId**() **const**
Get the Id of the zim file.

Returns The uuid stored in the zim file.

Entry **getRandomPage**() **const**
Get a random page.

Returns A random *Entry*. The entry is picked from all entries in the 'A' namespace. The main entry is excluded from the potential results.

Entry **getMainPage**() **const**
Get the entry of the main page.

Returns *Entry* of the main page as specified in the zim file.

bool **getMetadata**(**const** string &*name*, string &*value*) **const**
Get the content of a metadata.

Parameters

- **name** – **[in]** The name of the metadata.
- **value** – **[out]** The value will be set to the content of the metadata.

Returns True if it was possible to get the content of the metadata.

string **getName**() **const**
Get the name of the zim file.

Returns The name of the zim file as specified in the zim metadata.

string **getTitle**() **const**
Get the title of the zim file.

Returns The title of zim file as specified in the zim metadata. If no title has been set, return a title computed from the file path.

string **getCreator**() **const**
Get the creator of the zim file.

Returns The creator of the zim file as specified in the zim metadata.

string **getPublisher**() **const**
Get the publisher of the zim file.

Returns The publisher of the zim file as specified in the zim metadata.

string **getDate**() **const**
Get the date of the zim file.

Returns The date of the zim file as specified in the zim metadata.

string **getDescription**() **const**
Get the description of the zim file.

Returns The description of the zim file as specified in the zim metadata. If no description has been set, return the subtitle.

string **getLongDescription**() **const**
Get the long description of the zim file.

Returns The long description of the zim file as specifed in the zim metadata.

string **getLanguage**() **const**
Get the language of the zim file.

Returns The language of the zim file as specified in the zim metadata.

string **getLicense**() **const**
Get the license of the zim file.

Returns The license of the zim file as specified in the zim metadata.

string **getTags**(bool *original* = false) **const**
Get the tags of the zim file.

Parameters **original** – If true, return the original tags as specified in the zim metadata. Else, try to convert it to the new 'normalized' format.

Returns The tags of the zim file.

string **getTagStr**(**const** std::string &*tagName*) **const**
Get the value (as a string) of a specific tag.

According to https://wiki.openzim.org/wiki/Tags

Throws **std::out_of_range** – if the specified tag is not found.

Returns The value of the specified tag.

bool **getTagBool**(**const** std::string &*tagName*) **const**
Get the boolean value of a specific tag.

According to https://wiki.openzim.org/wiki/Tags

Throws **std::out_of_range** – if the specified tag is not found. std::domain_error if the value of the tag cannot be convert to bool.

Returns The boolean value of the specified tag.

string **getRelation**() **const**
Get the relations of the zim file.

Returns The relation of the zim file as specified in the zim metadata.

string **getFlavour**() **const**
Get the flavour of the zim file.

Returns The flavour of the zim file as specified in the zim metadata.

string **getSource**() **const**
Get the source of the zim file.

Returns The source of the zim file as specified in the zim metadata.

string **getScraper**() **const**
Get the scraper of the zim file.

Returns The scraper of the zim file as specified in the zim metadata.

bool **getFavicon**(string &*content*, string &*mimeType*) **const**
Get the favicon of the zim file.

Parameters

- **content** – **[out]** The content of the favicon.
- **mimeType** – **[out]** The mimeType of the favicon.

Returns True if a favicon has been found.

Entry **getEntryFromPath**(**const** std::string &*path*) **const**
Get an entry associated to an path.

Parameters **path** – The path of the entry.

Throws ***NoEntry*** – If no entry correspond to the path.

Returns The entry.

Entry **getEntryFromEncodedPath**(**const** std::string &*path*) **const**
Get an entry associated to an url encoded path.

Equivalent to `getEntryFromPath(urlDecode(path));`

Parameters **path** – The url encoded path.

Throws ***NoEntry*** – If no entry correspond to the path.

Returns The entry.

Entry **getEntryFromTitle**(**const** std::string &*title*) **const**
Get un entry associated to a title.

Parameters **title** – The title.

Returns The entry throw *NoEntry* If no entry correspond to the url.

bool **searchSuggestions**(**const** string &*prefix*, unsigned int *suggestionsCount*, **const** bool *reset* = true)

Search for entries with title starting with prefix (case sensitive).

Suggestions are stored in an internal vector and can be retrieved using `getNextSuggestion` method. This method is not thread safe and is deprecated. Use : bool searchSuggestions(const string& prefix, unsigned int suggestionsCount, SuggestionsList_t& results);

Parameters

- **prefix** – The prefix to search.
- **suggestionsCount** – How many suggestions to search for.
- **reset** – If true, remove previous suggestions in the internal vector. If false, add suggestions to the internal vector (until internal vector size is suggestionCount (or no more suggestion))

Returns True if some suggestions have been added to the internal vector.

bool **searchSuggestions**(**const** string &*prefix*, unsigned int *suggestionsCount*, *SuggestionsList_t* &*resuls*)

Search for entries with title starting with prefix (case sensitive).

Suggestions are added to the `result` vector.

Parameters

- **prefix** – The prefix to search.
- **suggestionsCount** – How many suggestions to search for.
- **result** – The vector where to store the suggestions.

Returns True if some suggestions have been added to the vector.

bool **searchSuggestionsSmart**(**const** string &*prefix*, unsigned int *suggestionsCount*)

Search for entries for the given prefix.

If the zim file has a internal fulltext index, the suggestions will be searched using it. Else the suggestions will be search using `searchSuggestions` while trying to be smart about case sensitivity (using `getTitleVariants`).

In any case, suggestions are stored in an internal vector and can be retrieved using `getNextSuggestion` method. The internal vector will be reset. This method is not thread safe and is deprecated. Use : bool searchSuggestionsSmart(const string& prefix, unsigned int suggestionsCount, SuggestionsList_t& results);

Parameters

- **prefix** – The prefix to search for.
- **suggestionsCount** – How many suggestions to search for.

bool **searchSuggestionsSmart**(**const** string &*prefix*, unsigned int *suggestionsCount*, *SuggestionsList_t* &*results*)

Search for entries for the given prefix.

If the zim file has a internal fulltext index, the suggestions will be searched using it. Else the suggestions will be search using `searchSuggestions` while trying to be smart about case sensitivity (using `getTitleVariants`).

In any case, suggestions are stored in an internal vector and can be retrieved using `getNextSuggestion` method. The internal vector will be reset.

Parameters

- **prefix** – The prefix to search for.
- **suggestionsCount** – How many suggestions to search for.
- **results** – The vector where to store the suggestions

Returns True if some suggestions have been added to the results.

bool **pathExists**(**const** string &*path*) **const**
Check if the path exists in the zim file.

Parameters **path** – the path to check.

Returns True if the path exists in the zim file.

bool **hasFulltextIndex**() **const**
Check if the zim file has a embedded fulltext index.

Returns True if the zim file has a embedded fulltext index and is not split (else the fulltext is not accessible).

std::vector<std::string> **getTitleVariants**(**const** std::string &*title*) **const**
Get potential case title variations for a title.

Parameters **title** – a title.

Returns the list of variantions.

bool **getNextSuggestion**(string &*title*)
Get the next suggestion title.

Parameters **title** – **[out]** the title of the suggestion.

Returns True if title has been set.

bool **getNextSuggestion**(string &*title*, string &*url*)
Get the next suggestion title and url.

Parameters

- **title** – **[out]** the title of the suggestion.
- **url** – **[out]** the url of the suggestion.

Returns True if title and url have been set.

bool **canCheckIntegrity**() **const**
Get if we can check zim file integrity (has a checksum).

Returns True if zim file have a checksum.

bool **isCorrupted**() **const**
Check is zim file is corrupted.

Returns True if zim file is corrupted.

unsigned int **getFileSize**() **const**
Return the total size of the zim file.

If zim file is split, return the sum of all parts' size.

Returns Size of the size file is KiB.

zim::Archive ***getZimArchive**() **const**
Get the zim file handler.

Returns The libzim file handler.

Protected Attributes

std::shared_ptr<zim::Archive> **zimArchive**

std::string **zimFilePath**

SuggestionsList_t **suggestions**

SuggestionsList_t::iterator **suggestionsOffset**

Class Result

- Defined in file_include_searcher.h

Class Documentation

class kiwix::**Result**

Public Functions

inline virtual ~Result()

virtual std::string **get_url**() = 0

virtual std::string **get_title**() = 0

virtual int **get_score**() = 0

virtual std::string **get_snippet**() = 0

virtual std::string **get_content**() = 0

virtual int **get_wordCount**() = 0

virtual int **get_size**() = 0

virtual std::string **get_zimId**() = 0

Class Searcher

- Defined in file_include_searcher.h

Class Documentation

class kiwix::**Searcher**

The *Searcher* class is reponsible to do different kind of search using the fulltext index.

Public Functions

Searcher()
The default constructor.

~Searcher()

bool **add_reader**(*Reader* *reader)
Add a reader (containing embedded fulltext index) to the search.

Parameters **reader** – The *Reader* for the zim containing the fulltext index.

Returns true if the reader has been added. false if the reader cannot be added (no embedded fulltext index present)

Reader ***get_reader**(int *index*)

void **search**(**const** std::string &*search*, unsigned int *resultStart*, unsigned int *maxResultCount*, **const** bool *verbose* = false)
Start a search on the zim associated to the *Searcher*.

Search results should be retrived using the getNextResult method.

Parameters

- **search** – The search query.
- **resultStart** – the start offset of the search results (used for pagination).
- **maxResultCount** – Maximum results to get from start (used for pagination).
- **verbose** – print some info on stdout if true.

void **geo_search**(float *latitude*, float *longitude*, float *distance*, unsigned int *resultStart*, unsigned int *maxResultCount*, **const** bool *verbose* = false)
Start a geographique search. The search return result for entry in a disc of center latitude/longitude and radius distance.

Search results should be retrived using the getNextResult method.

Parameters

- **latitude** – The latitude of the center point.
- **longitude** – The longitude of the center point.
- **distance** – The radius of the disc.
- **resultStart** – the start offset of the search results (used for pagination).
- **maxResultCount** – Maximum number of results to get from start (used for pagination).
- **verbose** – print some info on stdout if true.

void **suggestions**(std::string &*search*, **const** bool *verbose* = false)
Start a suggestion search. The search made depend of the "version" of the embedded index.

- If the index is newer enough and have a title namespace, the search is made in the titles only.
- Else the search is made on the whole article content. In any case, the search is made "partial" (as adding '*' at the end of the query)

Parameters

- **search** – The search query.
- **verbose** – print some info on stdout if true.

Result ***getNextResult*** ()
Get the next result of a started search. This is the method to use to loop hover the search results.

void ***restart_search*** ()
Restart the previous search. Next call to getNextResult will return the first result.

unsigned int ***getEstimatedResultCount*** ()
Get a estimation of the result count.

zim::SearchResultSet ***getSearchResultSet*** ()
Get a SearchResultSet object for current search

inline unsigned int ***getResultStart*** ()

inline unsigned int ***getMaxResultCount*** ()

Protected Functions

std::string ***beautifyInteger*** (***const*** unsigned int *number*)

void ***closeIndex*** ()

void ***searchInIndex*** (string *&search*, ***const*** unsigned int *resultStart*, ***const*** unsigned int *maxResultCount*, ***const*** bool *verbose* = false)

Protected Attributes

std::vector<*Reader**> ***readers***

std::unique_ptr<SearcherInternal> ***internal***

std::unique_ptr<SuggestionInternal> ***suggestionInternal***

std::string ***searchPattern***

unsigned int ***estimatedResultCount***

unsigned int ***resultStart***

unsigned int ***maxResultCount***

Class SearchRenderer

- Defined in file_include_search_renderer.h

Class Documentation

class kiwix::***SearchRenderer***
The SearcherRenderer class is used to render a search result to a html page.

Public Functions

SearchRenderer(*Searcher* *searcher, *NameMapper* *mapper)
The default constructor.

Parameters **humanReadableName** – The global zim's humanReadableName. Used to generate pagination links.

SearchRenderer(zim::SearchResultSet *srs*, *NameMapper* *mapper, unsigned int *start*, unsigned int *estimatedResultCount*)

~SearchRenderer()

void **setSearchPattern**(**const** std::string &*pattern*)

void **setSearchContent**(**const** std::string &*name*)
Set the search content id.

void **setProtocolPrefix**(**const** std::string &*prefix*)
Set protocol prefix.

void **setSearchProtocolPrefix**(**const** std::string &*prefix*)
Set search protocol prefix.

inline void **setPageLength**(unsigned int *pageLength*)
set result count per page

std::string **getHtml**()
Generate the html page with the resutls of the search.

Protected Functions

std::string **beautifyInteger**(**const** unsigned int *number*)

Protected Attributes

zim::SearchResultSet **m_srs**

NameMapper ***mp_nameMapper**

std::string **searchContent**

std::string **searchPattern**

std::string **protocolPrefix**

std::string **searchProtocolPrefix**

unsigned int **pageLength**

unsigned int **estimatedResultCount**

unsigned int **resultStart**

Class Server

- Defined in file_include_server.h

Class Documentation

class kiwix::**Server**

Public Functions

Server(*Library* *library, *NameMapper* *nameMapper = nullptr)
The default constructor.

Parameters **library** – The library to serve.

virtual **~Server**()

bool **start**()
Serve the content.

void **stop**()
Stop the daemon.

void **setRoot**(**const** std::string &*root*)

inline void **setAddress**(**const** std::string &*addr*)

inline void **setPort**(int *port*)

inline void **setNbThreads**(int *threads*)

inline void **setVerbose**(bool *verbose*)

inline void **setIndexTemplateString**(**const** std::string &*indexTemplateString*)

inline void **setTaskbar**(bool *withTaskbar*, bool *withLibraryButton*)

inline void **setBlockExternalLinks**(bool *blockExternalLinks*)

Protected Attributes

Library ***mp_library**

NameMapper ***mp_nameMapper**

std::string **m_root** = ""

std::string **m_addr** = ""

std::string **m_indexTemplateString** = ""

int **m_port** = 80

int **m_nbThreads** = 1

bool **m_verbose** = false

bool **m_withTaskbar** = true

bool **m_withLibraryButton** = true

bool **m_blockExternalLinks** = false

std::unique_ptr<InternalServer> **mp_server**

Class SuggestionItem

- Defined in file_include_reader.h

Class Documentation

class kiwix::**SuggestionItem**

The *SuggestionItem* is a helper class that contains the info about a single suggestion item.

Public Functions

inline explicit SuggestionItem(**const** std::string &*title*, **const** std::string &*normalizedTitle*, **const** std::string &*path*, **const** std::string &*snippet* = "")

inline const std::string &**getTitle**() **const**

inline const std::string &**getNormalizedTitle**() **const**

inline const std::string &**getPath**() **const**

inline const std::string &**getSnippet**() **const**

inline bool **hasSnippet**() **const**

Class UpdatableNameMapper

- Defined in file_include_name_mapper.h

Inheritance Relationships

Base Type

- public kiwix::NameMapper (*Class NameMapper*)

Class Documentation

class kiwix::**UpdatableNameMapper** : **public** kiwix::*NameMapper*

Public Functions

UpdatableNameMapper(*Library* &*library*, bool *withAlias*)

virtual std::string **getNameForId**(**const** std::string &*id*) **const**

virtual std::string **getIdForName**(**const** std::string &*name*) **const**

void **update**()

2.3.3 Enums

Enum supportedListMode

- Defined in file_include_library.h

Enum Documentation

enum kiwix::**supportedListMode**
Values:

enumerator ALL

enumerator LOCAL

enumerator REMOTE

enumerator NOLOCAL

enumerator NOREMOTE

enumerator VALID

enumerator NOVALID

Enum supportedListSortBy

- Defined in file_include_library.h

Enum Documentation

enum kiwix::**supportedListSortBy**
Values:

enumerator UNSORTED

enumerator TITLE

enumerator SIZE

enumerator DATE

enumerator CREATOR

enumerator PUBLISHER

2.3.4 Functions

Function kiwix::appendToDirectory

- Defined in file_include_tools.h

Function Documentation

std::string kiwix::**appendToDirectory**(**const** std::string &*basePath*, **const** std::string &*relativePath*)

Append a path to another one.

This function is provided as a small helper. It is probably better to use native tools to manipulate paths.

Parameters

- **basePath** – the base path.
- **relativePath** – a path to add to the base path, must be a relative path.

Returns The concatenation of the paths, using the right separator.

Function kiwix::computeAbsolutePath

- Defined in file_include_tools.h

Function Documentation

std::string kiwix::**computeAbsolutePath**(**const** std::string &*path*, **const** std::string &*relativePath*)

Compute the absolute path of a relative path based on another one

Equivalent to appendToDirectory followed by a normalization of the path.

This function is provided as a small helper. It is probably better to use native tools to manipulate paths.

Parameters

- **path** – the base path (if empty, current directory is taken).
- **relativePath** – the relative path.

Returns a absolute path.

Function kiwix::computeRelativePath

- Defined in file_include_tools.h

Function Documentation

std::string kiwix::**computeRelativePath**(**const** std::string &*path*, **const** std::string &*absolutePath*)

Compute the relative path of a path relative to another one

This function is provided as a small helper. It is probably better to use native tools to manipulate paths.

Parameters

- **path** – the base path.
- **absolutePath** – the absolute path to find the relative path for.

Returns a relative path (pointing to absolutePath, relative to path).

Function kiwix::converta2toa3

- Defined in file_include_tools.h

Function Documentation

std::string `kiwix::converta2toa3`(**const** std::string &*a2code*)
Convert language code from iso2 code to iso3

This function is provided as a small helper. It is probably better to use native tools to manipulate locales.

Parameters **a2code** – a iso2 code string.

Throws **std::out_of_range** – if iso2 code is not known.

Returns the corresponding iso3 code.

Function kiwix::fileExists

- Defined in file_include_tools.h

Function Documentation

bool `kiwix::fileExists`(**const** std::string &*path*)
checks if file exists.

This function returns boolean stating if file exists or not.

Parameters **path** – The absolute path provided in string format.

Returns Boolean representing if file exists or not.

Function kiwix::getCurrentDirectory

- Defined in file_include_tools.h

Function Documentation

std::string `kiwix::getCurrentDirectory`()
Return the current directory.

Returns the current directory (utf8 encoded)

Function kiwix::getDataDirectory

- Defined in file_include_tools.h

Function Documentation

std::string `kiwix::`**`getDataDirectory`**`()`
Return the data directory.

The data directory is a directory where to put data (zim files, ...) It depends of the platform and it may be changed by user using environment variable.

The resolution order is :

- `KIWIX_DATA_DIR` env variable (if set).
- On Windows : . `$APPDATA/kiwix` if $APPDATA is set . `$USERPROFILE/kiwix` if $USERPROFILE is set
- Else : . `$XDG_DATA_HOME/kiwix`if $XDG_DATA_HOME is set . `$HOME/.local/share/kiwx` if $HOWE is set
- current directory

Returns the path of the data directory (utf8 encoded)

Function kiwix::getExecutablePath

- Defined in file_include_tools.h

Function Documentation

std::string `kiwix::`**`getExecutablePath`**`(`bool *realPathOnly* = false`)`
Return the path of the executable

Some application may be packaged in auto extractible archive (Appimage) and the real executable is different of the path of the archive. If `realPathOnly` is true, return the path of the real executable instead of the archive launched by the user.

Parameters **`realPathOnly`** – If we must return the real path of the executable.

Returns the path of the executable (utf8 encoded)

Function kiwix::getFileContent

- Defined in file_include_tools.h

Function Documentation

std::string `kiwix::`**`getFileContent`**`(`**`const`** std::string &*path*`)`
Extracts content from given file.

This function provides content of a file provided it's path.

Parameters **`path`** – The absolute path provided in string format.

Returns Content of corresponding file in string format.

Function kiwix::getLastPathElement

- Defined in file_include_tools.h

Function Documentation

std::string `kiwix::`**`getLastPathElement`**(**`const`** std::string &*path*)

Get the last element of a path.

This function is provided as a small helper. It is probably better to use native tools to manipulate paths.

Parameters **`path`** – a path.

Returns The base name of the path or empty string if none (ending with a separator).

Function kiwix::getMimeTypeForFile

- Defined in file_include_tools.h

Function Documentation

std::string `kiwix::`**`getMimeTypeForFile`**(**`const`** std::string &*filename*)

provides mimetype from filename.

This function provides mimetype from file-name.

Parameters **`filename`** – string containing filename.

Returns mimetype from filename in string format.

Function kiwix::getSearchUrl

- Defined in file_include_opds_catalog.h

Function Documentation

std::string `kiwix::`**`getSearchUrl`**(**`const`** *Filter* &*f*)

Function kiwix::isRelativePath

- Defined in file_include_tools.h

Function Documentation

bool `kiwix::isRelativePath`(**const** std::string &*path*)
Tell if the path is a relative path.

This function is provided as a small helper. It is probably better to use native tools to manipulate paths.

Parameters **path** – A utf8 encoded path.

Returns true if the path is relative.

Function kiwix::removeLastPathElement

- Defined in file_include_tools.h

Function Documentation

std::string `kiwix::removeLastPathElement`(**const** std::string &*path*)
Remove the last element of a path.

This function is provided as a small helper. It is probably better to use native tools to manipulate paths.

Parameters **path** – a path.

Returns The parent directory (or empty string if none).

Function kiwix::sleep

- Defined in file_include_tools.h

Function Documentation

void `kiwix::sleep`(unsigned int *milliseconds*)
Sleep the current thread.

This function is provided as a small helper. It is probably better to use native tools.

Parameters **milliseconds** – The number of milliseconds to wait for.

Function kiwix::split

- Defined in file_include_tools.h

Function Documentation

std::vector<std::string> `kiwix::split`(**const** std::string &*str*, **const** std::string &*delims*, bool *dropEmpty* = true, bool *keepDelim* = false)
Split a string

This function is provided as a small helper. It is probably better to use native tools.

Assuming text = "foo:;bar;baz,oups;"

split(text, “:;”, true, true) => [“foo”, “:”, “;”, “bar”, “;”, “baz,oups”, “;”] split(text, “:;”, true, false) => [“foo”, “bar”, “baz,oups”] (default) split(text, “:;”, false, true) => [“foo”, “:”, “”, “;”, “bar”, “;”, “baz,oups”, “;”, “”] split(text, “:;”, false, false) => [“foo”, “”, “bar”, “baz,oups”, “”]

Parameters

- **str** – The string to split.
- **delims** – A string of potential delimiters. Each charater in the string can be a individual delimiters.
- **dropEmpty** – true if empty part must be dropped from the result.
- **keepDelim** – true if delimiter must be included from the result.

Returns a list of part (potentially containing delimiters)

2.3.5 Defines

Define DEPRECATED

- Defined in file_include_common.h

Define Documentation

DEPRECATED

Define KIWIX_LIBRARY_VERSION

- Defined in file_include_library.h

Define Documentation

KIWIX_LIBRARY_VERSION

2.3.6 Typedefs

Typedef kiwix::offset_type

- Defined in file_include_common.h

Typedef Documentation

typedef zim::offset_type `kiwix::`**offset_type**

Typedef kiwix::size_type

- Defined in file_include_common.h

Typedef Documentation

typedef zim::size_type kiwix::**size_type**

Typedef kiwix::SuggestionsList_t

- Defined in file_include_reader.h

Typedef Documentation

using kiwix::**SuggestionsList_t** = std::vector<*SuggestionItem*>

The *Reader* class is the class who allow to get an entry content from a zim file.

INDEX