

MLBookProc

1.1

Generated by Doxygen 1.13.2



<b>1 MLBookProc</b>	<b>1</b>
<b>2 Hierarchical Index</b>	<b>3</b>
2.1 Class Hierarchy	3
<b>3 Class Index</b>	<b>5</b>
3.1 Class List	5
<b>4 Class Documentation</b>	<b>7</b>
4.1 AddBook Class Reference	7
4.1.1 Detailed Description	8
4.1.2 Constructor & Destructor Documentation	8
4.1.2.1 AddBook()	8
4.1.3 Member Function Documentation	8
4.1.3.1 add_to_existing_archive()	8
4.1.3.2 add_to_existing_archive_dir()	8
4.1.3.3 archive_filenames()	9
4.1.3.4 overwrite_archive()	9
4.1.3.5 overwrite_archive_dir()	9
4.1.3.6 simple_add()	10
4.1.3.7 simple_add_dir()	10
4.2 ArchEntry Class Reference	10
4.2.1 Detailed Description	11
4.3 ArchiveFileEntry Class Reference	11
4.3.1 Detailed Description	12
4.4 ArchiveRemoveEntry Class Reference	12
4.4.1 Detailed Description	13
4.5 ARCHParser Class Reference	13
4.5.1 Detailed Description	15
4.5.2 Constructor & Destructor Documentation	15
4.5.2.1 ARCHParser()	15
4.5.3 Member Function Documentation	15
4.5.3.1 arch_parser()	15
4.6 AuxFunc Class Reference	16
4.6.1 Detailed Description	17
4.6.2 Member Function Documentation	17
4.6.2.1 copy_book_callback()	17
4.6.2.2 create()	18
4.6.2.3 detect_encoding()	18
4.6.2.4 get_activated()	18
4.6.2.5 get_charset_conv_quantity()	18
4.6.2.6 get_converter_by_number()	18
4.6.2.7 get_extension()	19

4.6.2.8	<a href="#">get_genre_list()</a>	19
4.6.2.9	<a href="#">get_selfpath()</a>	19
4.6.2.10	<a href="#">get_supported_archive_types_packing()</a>	20
4.6.2.11	<a href="#">get_supported_archive_types_unpacking()</a>	20
4.6.2.12	<a href="#">get_supported_types()</a>	20
4.6.2.13	<a href="#">getDJVUContext()</a>	20
4.6.2.14	<a href="#">homePath()</a>	20
4.6.2.15	<a href="#">html_to_utf8()</a>	20
4.6.2.16	<a href="#">if_supported_type()</a>	21
4.6.2.17	<a href="#">libgcrypt_error_handling()</a>	21
4.6.2.18	<a href="#">open_book_callback()</a>	21
4.6.2.19	<a href="#">randomFileName()</a>	22
4.6.2.20	<a href="#">share_path()</a>	22
4.6.2.21	<a href="#">stringToLower()</a>	22
4.6.2.22	<a href="#">temp_path()</a>	22
4.6.2.23	<a href="#">time_t_to_date()</a>	22
4.6.2.24	<a href="#">to_hex()</a>	23
4.6.2.25	<a href="#">to_utf_8()</a>	23
4.6.2.26	<a href="#">utf8_to_system()</a>	23
4.6.2.27	<a href="#">utf_8_to()</a>	24
4.7	<a href="#">BaseKeeper Class Reference</a>	24
4.7.1	<a href="#">Detailed Description</a>	25
4.7.2	<a href="#">Constructor &amp; Destructor Documentation</a>	25
4.7.2.1	<a href="#">BaseKeeper()</a>	25
4.7.3	<a href="#">Member Function Documentation</a>	25
4.7.3.1	<a href="#">booksWithNotes()</a>	25
4.7.3.2	<a href="#">collectionAuthors()</a>	26
4.7.3.3	<a href="#">get_base_vector()</a>	26
4.7.3.4	<a href="#">get_books_path()</a>	26
4.7.3.5	<a href="#">loadCollection()</a>	26
4.7.3.6	<a href="#">searchBook()</a>	27
4.7.4	<a href="#">Member Data Documentation</a>	27
4.7.4.1	<a href="#">auth_show_progr</a>	27
4.8	<a href="#">BookBaseEntry Class Reference</a>	27
4.8.1	<a href="#">Detailed Description</a>	28
4.8.2	<a href="#">Constructor &amp; Destructor Documentation</a>	28
4.8.2.1	<a href="#">BookBaseEntry()</a>	28
4.9	<a href="#">BookInfo Class Reference</a>	29
4.9.1	<a href="#">Detailed Description</a>	29
4.9.2	<a href="#">Constructor &amp; Destructor Documentation</a>	29
4.9.2.1	<a href="#">BookInfo()</a>	29
4.9.3	<a href="#">Member Function Documentation</a>	29

4.9.3.1 <a href="#">get_book_info()</a>	29
4.9.3.2 <a href="#">set_dpi()</a>	30
4.10 <a href="#">BookInfoEntry Class Reference</a>	30
4.10.1 <a href="#">Detailed Description</a>	31
4.10.2 <a href="#">Member Enumeration Documentation</a>	31
4.10.2.1 <a href="#">cover_types</a>	31
4.11 <a href="#">BookMarks Class Reference</a>	32
4.11.1 <a href="#">Detailed Description</a>	32
4.11.2 <a href="#">Constructor &amp; Destructor Documentation</a>	32
4.11.2.1 <a href="#">BookMarks()</a>	32
4.11.3 <a href="#">Member Function Documentation</a>	33
4.11.3.1 <a href="#">createBookMark()</a>	33
4.11.3.2 <a href="#">getBookMarks()</a>	33
4.11.3.3 <a href="#">removeBookMark()</a>	33
4.12 <a href="#">BookParseEntry Class Reference</a>	33
4.12.1 <a href="#">Detailed Description</a>	34
4.12.2 <a href="#">Member Data Documentation</a>	34
4.12.2.1 <a href="#">book_genre</a>	34
4.12.2.2 <a href="#">book_name</a>	35
4.12.2.3 <a href="#">book_path</a>	35
4.13 <a href="#">ByteOrder Class Reference</a>	35
4.13.1 <a href="#">Detailed Description</a>	37
4.13.2 <a href="#">Constructor &amp; Destructor Documentation</a>	37
4.13.2.1 <a href="#">ByteOrder()</a> [1/8]	37
4.13.2.2 <a href="#">ByteOrder()</a> [2/8]	37
4.13.2.3 <a href="#">ByteOrder()</a> [3/8]	37
4.13.2.4 <a href="#">ByteOrder()</a> [4/8]	37
4.13.2.5 <a href="#">ByteOrder()</a> [5/8]	38
4.13.2.6 <a href="#">ByteOrder()</a> [6/8]	38
4.13.2.7 <a href="#">ByteOrder()</a> [7/8]	38
4.13.2.8 <a href="#">ByteOrder()</a> [8/8]	38
4.13.3 <a href="#">Member Function Documentation</a>	39
4.13.3.1 <a href="#">get_big()</a>	39
4.13.3.2 <a href="#">get_little()</a>	39
4.13.3.3 <a href="#">get_native()</a>	39
4.13.3.4 <a href="#">operator=()</a> [1/8]	39
4.13.3.5 <a href="#">operator=()</a> [2/8]	40
4.13.3.6 <a href="#">operator=()</a> [3/8]	40
4.13.3.7 <a href="#">operator=()</a> [4/8]	40
4.13.3.8 <a href="#">operator=()</a> [5/8]	40
4.13.3.9 <a href="#">operator=()</a> [6/8]	41
4.13.3.10 <a href="#">operator=()</a> [7/8]	41

4.13.3.11 operator=() [8/8]	41
4.13.3.12 set_big()	41
4.13.3.13 set_little()	42
4.14 CreateCollection Class Reference	42
4.14.1 Detailed Description	44
4.14.2 Constructor & Destructor Documentation	44
4.14.2.1 CreateCollection() [1/2]	44
4.14.2.2 CreateCollection() [2/2]	45
4.14.3 Member Function Documentation	45
4.14.3.1 closeBaseFile()	45
4.14.3.2 createCollection()	45
4.14.3.3 openBaseFile()	46
4.14.3.4 threadRegulator()	46
4.14.3.5 write_file_to_base()	46
4.14.4 Member Data Documentation	46
4.14.4.1 already_hashed	46
4.14.4.2 base_path	47
4.14.4.3 books_path	47
4.14.4.4 current_bytes	47
4.14.4.5 need_to_parse	47
4.14.4.6 progress	47
4.14.4.7 pulse	48
4.14.4.8 rar_support	48
4.14.4.9 signal_total_bytes	48
4.15 DCPParser Class Reference	48
4.15.1 Detailed Description	50
4.15.2 Constructor & Destructor Documentation	50
4.15.2.1 DCPParser()	50
4.15.3 Member Function Documentation	50
4.15.3.1 dcAuthor()	50
4.15.3.2 dcDate()	50
4.15.3.3 dcDescription()	51
4.15.3.4 dcGenre()	51
4.15.3.5 dcIdentifier()	51
4.15.3.6 dcLanguage()	52
4.15.3.7 dcPublisher()	52
4.15.3.8 dcSource()	52
4.15.3.9 dcTitle()	53
4.16 DJVUParser Class Reference	53
4.16.1 Detailed Description	53
4.16.2 Constructor & Destructor Documentation	54
4.16.2.1 DJVUParser()	54

4.16.3 Member Function Documentation	54
4.16.3.1 djvu_book_info()	54
4.16.3.2 djvu_parser()	54
4.17 ElectroBookInfoEntry Class Reference	55
4.17.1 Detailed Description	55
4.18 EPUBParser Class Reference	56
4.18.1 Detailed Description	58
4.18.2 Constructor & Destructor Documentation	58
4.18.2.1 EPUBParser()	58
4.18.3 Member Function Documentation	59
4.18.3.1 epub_book_info()	59
4.18.3.2 epub_parser()	59
4.19 FB2Parser Class Reference	59
4.19.1 Detailed Description	60
4.19.2 Constructor & Destructor Documentation	61
4.19.2.1 FB2Parser()	61
4.19.3 Member Function Documentation	61
4.19.3.1 fb2_book_info()	61
4.19.3.2 fb2_parser()	61
4.20 FileParseEntry Class Reference	62
4.20.1 Detailed Description	62
4.20.2 Member Data Documentation	62
4.20.2.1 books	62
4.20.2.2 file_hash	63
4.20.2.3 file_rel_path	63
4.21 FormatAnnotation Class Reference	63
4.21.1 Detailed Description	64
4.21.2 Constructor & Destructor Documentation	64
4.21.2.1 FormatAnnotation()	64
4.21.3 Member Function Documentation	64
4.21.3.1 final_cleaning()	64
4.21.3.2 remove_escape_sequences()	65
4.21.3.3 removeAllTags()	65
4.21.3.4 replace_tags()	65
4.21.3.5 setTagReplacementTable()	65
4.22 Genre Class Reference	66
4.22.1 Detailed Description	66
4.22.2 Member Data Documentation	66
4.22.2.1 genre_code	66
4.22.2.2 genre_name	67
4.23 GenreGroup Class Reference	67
4.23.1 Detailed Description	67

4.23.2 Member Data Documentation	68
4.23.2.1 group_name	68
4.24 Hasher Class Reference	68
4.24.1 Detailed Description	69
4.24.2 Constructor & Destructor Documentation	69
4.24.2.1 Hasher()	69
4.24.3 Member Function Documentation	69
4.24.3.1 buf_hashing()	69
4.24.3.2 file_hashing()	69
4.24.4 Member Data Documentation	70
4.24.4.1 cancel	70
4.24.4.2 stop_all_signal	70
4.25 LibArchive Class Reference	70
4.25.1 Detailed Description	72
4.25.2 Constructor & Destructor Documentation	72
4.25.2.1 LibArchive()	72
4.25.3 Member Function Documentation	72
4.25.3.1 createArchFile()	72
4.25.3.2 fileInfo()	73
4.25.3.3 fileNames()	73
4.25.3.4 fileNamesStream()	73
4.25.3.5 libarchive_error()	74
4.25.3.6 libarchive_packing() [1/2]	74
4.25.3.7 libarchive_packing() [2/2]	74
4.25.3.8 libarchive_read_entry()	75
4.25.3.9 libarchive_read_entry_str()	75
4.25.3.10 libarchive_read_init()	76
4.25.3.11 libarchive_read_init_fallback()	76
4.25.3.12 libarchive_remove_entry()	76
4.25.3.13 libarchive_remove_init()	76
4.25.3.14 libarchive_write_data()	77
4.25.3.15 libarchive_write_data_from_file()	77
4.25.3.16 libarchive_write_directory()	78
4.25.3.17 libarchive_write_file()	78
4.25.3.18 libarchive_write_init()	79
4.25.3.19 unpackByFileNameStream()	80
4.25.3.20 unpackByFileNameStreamStr()	80
4.25.3.21 unpackByPosition()	81
4.25.3.22 unpackByPositionStr()	81
4.25.3.23 writeToArchive()	82
4.26 MLEException Class Reference	82
4.26.1 Detailed Description	83



4.26.2 Constructor & Destructor Documentation	83
4.26.2.1 MLEException()	83
4.26.3 Member Function Documentation	83
4.26.3.1 what()	83
4.27 NotesBaseEntry Class Reference	83
4.27.1 Detailed Description	84
4.27.2 Constructor & Destructor Documentation	84
4.27.2.1 NotesBaseEntry()	84
4.28 NotesKeeper Class Reference	84
4.28.1 Detailed Description	85
4.28.2 Constructor & Destructor Documentation	85
4.28.2.1 NotesKeeper()	85
4.28.3 Member Function Documentation	86
4.28.3.1 editNote()	86
4.28.3.2 getNote()	86
4.28.3.3 getNotesForCollection()	86
4.28.3.4 loadBase()	87
4.28.3.5 readNote()	87
4.28.3.6 readNoteText()	87
4.28.3.7 refreshCollection()	87
4.28.3.8 removeCollection()	88
4.28.3.9 removeNotes()	88
4.29 ODTParser Class Reference	89
4.29.1 Detailed Description	91
4.29.2 Constructor & Destructor Documentation	91
4.29.2.1 ODTParser()	91
4.29.3 Member Function Documentation	92
4.29.3.1 odtBookInfo()	92
4.29.3.2 odtParser()	92
4.30 OmpLockGuard Class Reference	92
4.30.1 Detailed Description	93
4.30.2 Constructor & Destructor Documentation	93
4.30.2.1 OmpLockGuard()	93
4.31 OpenBook Class Reference	93
4.31.1 Detailed Description	94
4.31.2 Constructor & Destructor Documentation	94
4.31.2.1 OpenBook()	94
4.31.3 Member Function Documentation	94
4.31.3.1 open_book()	94
4.32 PaperBookInfoEntry Class Reference	95
4.32.1 Detailed Description	95
4.33 PDFParser Class Reference	95

4.33.1 Detailed Description	96
4.33.2 Constructor & Destructor Documentation	96
4.33.2.1 PDFParser()	96
4.33.3 Member Function Documentation	96
4.33.3.1 pdf_annotation_n_cover()	96
4.33.3.2 pdf_parser()	96
4.34 RefreshCollection Class Reference	97
4.34.1 Detailed Description	100
4.34.2 Constructor & Destructor Documentation	100
4.34.2.1 RefreshCollection()	100
4.34.3 Member Function Documentation	100
4.34.3.1 editBook()	100
4.34.3.2 refreshBook()	101
4.34.3.3 refreshCollection()	101
4.34.3.4 refreshFile()	101
4.34.3.5 set_rar_support()	101
4.34.4 Member Data Documentation	102
4.34.4.1 bytes_hashed	102
4.34.4.2 total_bytes_to_hash	102
4.35 RemoveBook Class Reference	102
4.35.1 Detailed Description	102
4.35.2 Constructor & Destructor Documentation	102
4.35.2.1 RemoveBook()	102
4.35.3 Member Function Documentation	103
4.35.3.1 removeBook()	103
4.36 ReplaceTagItem Class Reference	103
4.36.1 Detailed Description	104
4.37 SelfRemovingPath Class Reference	104
4.37.1 Detailed Description	104
4.37.2 Constructor & Destructor Documentation	104
4.37.2.1 SelfRemovingPath()	104
4.37.3 Member Function Documentation	105
4.37.3.1 operator=()	105
4.38 XMLParser Class Reference	105
4.38.1 Detailed Description	106
4.38.2 Constructor & Destructor Documentation	106
4.38.2.1 XMLParser()	106
4.38.3 Member Function Documentation	106
4.38.3.1 get_book_encoding()	106
4.38.3.2 get_element_attribute()	107
4.38.3.3 get_tag()	107
4.38.3.4 htmlSymbolsReplacement()	107

---

4.38.3.5 listAllTags()	108
4.38.3.6 removeAllTags()	108
4.38.3.7 searchTag()	108
4.39 XMLTag Class Reference	109
4.39.1 Detailed Description	109
4.39.2 Member Function Documentation	110
4.39.2.1 hasContent()	110
4.39.3 Member Data Documentation	110
4.39.3.1 content_end	110
4.39.3.2 content_start	110
4.39.3.3 element	110
<b>Index</b>	<b>111</b>



# Chapter 1

## MLBookProc

**MLBookProc** is a library for managing `.fb2`, `.epub`, `.pdf`, `.djvu` and `.odt` e-book file collections. It can also works with same formats packed in `zip`, `7z`, `jar`, `cpio`, `iso`, `tar`, `tar.gz`, `tar.bz2`, `tar.xz`, `rar` archives (`rar` archives are available only for reading) itself or packed in same types of archives with `.fbd` files (`epub`, `djvu` and `pdf` books). **MLBookProc** creates own database and does not change files content, names or location.

To start add cmake package MLBookProc to your project.

```
find_package(MLBookProc)
if(MLBookProc_FOUND)
    target_include_directories(myproject PRIVATE MLBookProc::mlbookproc)
    target_link_libraries(myproject PUBLIC MLBookProc::mlbookproc)
endif()
```

Then create [AuxFunc](#) object. Further reading: [CreateCollection](#), [RefreshCollection](#), [BaseKeeper](#), [BookMarks](#), [NotesKeeper](#), [BookInfo](#), [OpenBook](#).



## Chapter 2

# Hierarchical Index

### 2.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

AddBook . . . . .	7
ArchEntry . . . . .	10
ArchiveFileEntry . . . . .	11
ArchiveRemoveEntry . . . . .	12
AuxFunc . . . . .	16
BaseKeeper . . . . .	24
BookBaseEntry . . . . .	27
BookInfo . . . . .	29
BookInfoEntry . . . . .	30
BookMarks . . . . .	32
BookParseEntry . . . . .	33
ByteOrder . . . . .	35
DJVUParser . . . . .	53
ElectroBookInfoEntry . . . . .	55
FileParseEntry . . . . .	62
Genre . . . . .	66
GenreGroup . . . . .	67
Hasher . . . . .	68
CreateCollection . . . . .	42
RefreshCollection . . . . .	97
LibArchive . . . . .	70
ARCHParser . . . . .	13
EPUBParser . . . . .	56
ODTParser . . . . .	89
MLException . . . . .	82
NotesBaseEntry . . . . .	83
NotesKeeper . . . . .	84
OmpLockGuard . . . . .	92
OpenBook . . . . .	93
PaperBookInfoEntry . . . . .	95
PDFParser . . . . .	95
RemoveBook . . . . .	102
ReplaceTagItem . . . . .	103
SelfRemovingPath . . . . .	104
XMLParser . . . . .	105

DCParser . . . . .	48
EPUBParser . . . . .	56
FB2Parser . . . . .	59
FormatAnnotation . . . . .	63
ODTParser . . . . .	89
XMLTag . . . . .	109



## Chapter 3

# Class Index

### 3.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

<a href="#">AddBook</a>	
The <a href="#">AddBook</a> class	7
<a href="#">ArchEntry</a>	
The <a href="#">ArchEntry</a> class	10
<a href="#">ArchiveFileEntry</a>	
The <a href="#">ArchiveFileEntry</a> class	11
<a href="#">ArchiveRemoveEntry</a>	
The <a href="#">ArchiveRemoveEntry</a> class	12
<a href="#">ARCHParser</a>	
The <a href="#">ARCHParser</a> class	13
<a href="#">AuxFunc</a>	
The <a href="#">AuxFunc</a> class	16
<a href="#">BaseKeeper</a>	
The <a href="#">BaseKeeper</a> class	24
<a href="#">BookBaseEntry</a>	
The <a href="#">BookBaseEntry</a> class	27
<a href="#">BookInfo</a>	
The <a href="#">BookInfo</a> class	29
<a href="#">BookInfoEntry</a>	
The <a href="#">BookInfoEntry</a> class	30
<a href="#">BookMarks</a>	
The <a href="#">BookMarks</a> class	32
<a href="#">BookParseEntry</a>	
The <a href="#">BookParseEntry</a> class	33
<a href="#">ByteOrder</a>	
The <a href="#">ByteOrder</a> class	35
<a href="#">CreateCollection</a>	
The <a href="#">CreateCollection</a> class	42
<a href="#">DCParser</a>	
The <a href="#">DCParser</a> class	48
<a href="#">DJVUParser</a>	
The <a href="#">DJVUParser</a> class	53
<a href="#">ElectroBookInfoEntry</a>	
The <a href="#">ElectroBookInfoEntry</a> class	55
<a href="#">EPUBParser</a>	
The <a href="#">EPUBParser</a> class	56

<a href="#">FB2Parser</a>	
The <a href="#">FB2Parser</a> class	59
<a href="#">FileParseEntry</a>	
The <a href="#">FileParseEntry</a> class	62
<a href="#">FormatAnnotation</a>	
The <a href="#">FormatAnnotation</a> class	63
<a href="#">Genre</a>	
The <a href="#">Genre</a> class	66
<a href="#">GenreGroup</a>	
The <a href="#">GenreGroup</a> class	67
<a href="#">Hasher</a>	
The <a href="#">Hasher</a> class	68
<a href="#">LibArchive</a>	
The <a href="#">LibArchive</a> class	70
<a href="#">MLException</a>	
The <a href="#">MLException</a> class	82
<a href="#">NotesBaseEntry</a>	
The <a href="#">NotesBaseEntry</a> class	83
<a href="#">NotesKeeper</a>	
The <a href="#">NotesKeeper</a> class	84
<a href="#">ODTParser</a>	
The <a href="#">ODTParser</a> class	89
<a href="#">OmpLockGuard</a>	
The <a href="#">OmpLockGuard</a> class	92
<a href="#">OpenBook</a>	
The <a href="#">OpenBook</a> class	93
<a href="#">PaperBookInfoEntry</a>	
The <a href="#">PaperBookInfoEntry</a> class	95
<a href="#">PDFParser</a>	
The <a href="#">PDFParser</a> class	95
<a href="#">RefreshCollection</a>	
The <a href="#">RefreshCollection</a> class	97
<a href="#">RemoveBook</a>	
The <a href="#">RemoveBook</a> class	102
<a href="#">ReplaceTagItem</a>	
The <a href="#">ReplaceTagItem</a> class	103
<a href="#">SelfRemovingPath</a>	
The <a href="#">SelfRemovingPath</a> class	104
<a href="#">XMLParser</a>	
The <a href="#">XMLParser</a> class	105
<a href="#">XMLTag</a>	
The <a href="#">XMLTag</a> class	109

# Chapter 4

## Class Documentation

### 4.1 AddBook Class Reference

The [AddBook](#) class.

```
#include <AddBook.h>
```

#### Public Member Functions

- [AddBook](#) (const std::shared\_ptr< [AuxFunc](#) > &af, const std::string &collection\_name, const bool &remove\_←  
\_sources, const std::shared\_ptr< [BookMarks](#) > &bookmarks)  
*AddBook constructor.*
- void [simple\\_add](#) (const std::vector< std::tuple< std::filesystem::path, std::filesystem::path > > &books)  
*Adds single book files.*
- void [simple\\_add\\_dir](#) (const std::vector< std::tuple< std::filesystem::path, std::filesystem::path > > &books)  
*Adds directories containing book files to collection.*
- void [overwrite\\_archive](#) (const std::filesystem::path &archive\_path, const std::vector< std::tuple< std←  
::filesystem::path, std::filesystem::path > > &books)  
*Removes archive from collection and replaces it by new one.*
- void [overwrite\\_archive\\_dir](#) (const std::filesystem::path &archive\_path, const std::vector< std::tuple< std←  
::filesystem::path, std::filesystem::path > > &books)  
*Removes archive from collection and replaces it by new one.*
- void [add\\_to\\_existing\\_archive](#) (const std::filesystem::path &archive\_path, const std::vector< std::tuple< std←  
::filesystem::path, std::filesystem::path > > &books)  
*Adds books to existing archive.*
- void [add\\_to\\_existing\\_archive\\_dir](#) (const std::filesystem::path &archive\_path, const std::vector< std::tuple<←  
std::filesystem::path, std::filesystem::path > > &books)  
*Adds books to existing archive.*

#### Static Public Member Functions

- static std::vector< std::string > [archive\\_filenames](#) (const std::filesystem::path &archive\_path, const std←  
::shared\_ptr< [AuxFunc](#) > &af)  
*Lists all files in archive.*

### 4.1.1 Detailed Description

The [AddBook](#) class.

[AddBook](#) contains various methods to add books to collections.

### 4.1.2 Constructor & Destructor Documentation

#### 4.1.2.1 AddBook()

```
AddBook::AddBook (
    const std::shared_ptr< AuxFunc > & af,
    const std::string & collection_name,
    const bool & remove_sources,
    const std::shared_ptr< BookMarks > & bookmarks)
```

[AddBook](#) constructor.

Parameters

<i>af</i>	<a href="#">AuxFunc</a> object.
<i>collection_name</i>	name of collection book to be added to.
<i>remove_sources</i>	if <i>true</i> , <b>MLBookProc</b> will remove source file after book has been added.
<i>bookmarks</i>	<a href="#">BookMarks</a> object.

### 4.1.3 Member Function Documentation

#### 4.1.3.1 add\_to\_existing\_archive()

```
void AddBook::add_to_existing_archive (
    const std::filesystem::path & archive_path,
    const std::vector< std::tuple< std::filesystem::path, std::filesystem::path > >
    & books)
```

Adds books to existing archive.

Unpacks existing archive and packs it again with old and new books inside.

Parameters

<i>archive_path</i>	absolute path to archive in collection.
<i>books</i>	vector, containing absolute paths to source books files (first tuple element) and relative paths to books in archive (second tuple element).

#### 4.1.3.2 add\_to\_existing\_archive\_dir()

```
void AddBook::add_to_existing_archive_dir (
    const std::filesystem::path & archive_path,
    const std::vector< std::tuple< std::filesystem::path, std::filesystem::path > >
    & books)
```

Adds books to existing archive.

Same as [add\\_to\\_existing\\_archive\(\)](#), but adds directories to existing archive.

## Parameters

<i>archive_path</i>	absolute path to archive in collection.
<i>books</i>	vector, containing absolute paths to source directories (first tuple element) and relative paths to directories in archive (second tuple element).

**4.1.3.3 archive\_filenames()**

```
static std::vector< std::string > AddBook::archive_filenames (
    const std::filesystem::path & archive_path,
    const std::shared_ptr< AuxFunc > & af) [static]
```

Lists all files in archive.

## Parameters

<i>archive_path</i>	absolute path to archive.
<i>af</i>	smart pointer to <a href="#">AuxFunc</a> object.

## Returns

Vector containing relative paths of archive files.

**4.1.3.4 overwrite\_archive()**

```
void AddBook::overwrite_archive (
    const std::filesystem::path & archive_path,
    const std::vector< std::tuple< std::filesystem::path, std::filesystem::path > >
    & books)
```

Removes archive from collection and replaces it by new one.

Removes archive from collection (if it exists) and creates new archive with the same name, containing books from books vector.

## Parameters

<i>archive_path</i>	absolute path to archive file in collection.
<i>books</i>	vector, containing absolute paths to source books files (first tuple element) and relative paths to books files in archive (second tuple element).

**4.1.3.5 overwrite\_archive\_dir()**

```
void AddBook::overwrite_archive_dir (
    const std::filesystem::path & archive_path,
    const std::vector< std::tuple< std::filesystem::path, std::filesystem::path > >
    & books)
```

Removes archive from collection and replaces it by new one.

Same as [overwrite\\_archive\(\)](#), but creates new archive, containing directories from books vector.

## Parameters

<i>archive_path</i>	absolute path to archive file in collection.
<i>books</i>	vector, containing absolute paths to source directories (first tuple element) and relative paths to directories in archive (second tuple element).

**4.1.3.6 simple\_add()**

```
void AddBook::simple_add (
    const std::vector< std::tuple< std::filesystem::path, std::filesystem::path > >
    & books)
```

Adds single book files.

Adds books, listed in vector, to collection and parse them. User must set both paths manually.

## Warning

Second tuples paths must not be outside collection books directory.

## Parameters

<i>books</i>	vector, containing absolute paths to source files (first tuple element) and absolute paths of books files in collection (second tuple element).
--------------	---

**4.1.3.7 simple\_add\_dir()**

```
void AddBook::simple_add_dir (
    const std::vector< std::tuple< std::filesystem::path, std::filesystem::path > >
    & books)
```

Adds directories containing book files to collection.

Same as [simple\\_add\(\)](#), but adds directories containing books to collection.

## Warning

Second tuples paths must not be outside collection books directory.

## Parameters

<i>books</i>	vector, containing absolute paths to source directories (first tuple element) and absolute paths of directories in collection (second tuple element).
--------------	---

**4.2 ArchEntry Class Reference**

The [ArchEntry](#) class.

```
#include <ArchEntry.h>
```

**Public Member Functions**

- **ArchEntry** ()  
*ArchEntry* constructor.
- **ArchEntry** (const [ArchEntry](#) &other)  
*ArchEntry* copy constructor.
- **ArchEntry** ([ArchEntry](#) &&other)  
*ArchEntry* move constructor.
- [ArchEntry](#) & **operator=** (const [ArchEntry](#) &other)  
*operator =*
- [ArchEntry](#) & **operator=** ([ArchEntry](#) &&other)  
*operator =*

**Public Attributes**

- uint64\_t **size** = 0  
*Size of unpacked entry object (if available, 0 otherwise).*
- uint64\_t **compressed\_size** = 0  
*Size of compressed object (if available, 0 otherwise).*
- int64\_t **position** = -1  
*Position of entry in archive file (if available, -1 otherwise).*
- std::string **filename**  
*Path to file or directory in archive.*

**4.2.1 Detailed Description**

The [ArchEntry](#) class.

Auxiliary class for [LibArchive](#). Contains some technical info about compressed files and directories. In most cases you do not need to create ArchEntry objects yourself.

**4.3 ArchiveFileEntry Class Reference**

The [ArchiveFileEntry](#) class.

```
#include <ArchiveFileEntry.h>
```

**Public Member Functions**

- **ArchiveFileEntry** ()  
*ArchiveFileEntry* constructor.
- virtual ~**ArchiveFileEntry** ()  
*ArchiveFileEntry* destructor.

## Public Attributes

- `std::fstream file`  
*Archive file stream.*
- `std::filesystem::path file_path`  
*Archive file absolute path.*
- `la_ssize_t buf_sz = 1048576`  
*Size of buffer to be used in file stream.*
- `la_ssize_t read_bytes = 0`  
*Number of bites already read from archive file.*
- `la_ssize_t file_size = 0`  
*Archive file size.*
- `char * read_buf = nullptr`  
*Buffer to be used in file stream.*

### 4.3.1 Detailed Description

The [ArchiveFileEntry](#) class.

Auxiliary class for [LibArchive](#) methods. In most cases you do not need to create it directly, use [LibArchive::createArchFile\(\)](#) method instead.

## 4.4 ArchiveRemoveEntry Class Reference

The [ArchiveRemoveEntry](#) class.

```
#include <ArchiveRemoveEntry.h>
```

## Public Member Functions

- **ArchiveRemoveEntry ()**  
*ArchiveRemoveEntry constructor.*
- virtual **~ArchiveRemoveEntry ()**  
*ArchiveRemoveEntry destructor.*
- **ArchiveRemoveEntry** (const [ArchiveRemoveEntry](#) &other)  
*ArchiveRemoveEntry copy constructor.*
- **ArchiveRemoveEntry** ([ArchiveRemoveEntry](#) &&other)  
*ArchiveRemoveEntry move constructor.*
- **ArchiveRemoveEntry & operator=** (const [ArchiveRemoveEntry](#) &other)  
*operator =*
- **ArchiveRemoveEntry & operator=** ([ArchiveRemoveEntry](#) &&other)  
*operator =*

## Public Attributes

- `std::shared_ptr< archive > a_read`  
*libarchive object, file to be removed from.*
- `std::shared_ptr< archive > a_write`  
*libarchive object for new archive.*
- `std::shared_ptr< ArchiveFileEntry > fl`  
*ArchiveFileEntry object, file to be removed from.*



### 4.4.1 Detailed Description

The [ArchiveRemoveEntry](#) class.

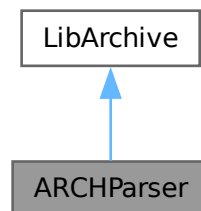
Auxiliary class for [LibArchive](#), to be used in case of removing files from archives. In most cases you do not need to use it directly.

## 4.5 ARCHParser Class Reference

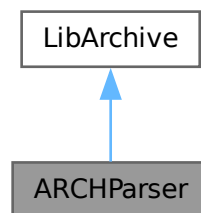
The [ARCHParser](#) class.

```
#include <ARCHParser.h>
```

Inheritance diagram for ARCHParser:



Collaboration diagram for ARCHParser:



### Public Member Functions

- [ARCHParser](#) (const std::shared\_ptr< [AuxFunc](#) > &af, const bool &rar\_support)  
*ARCHParser constructor.*
- virtual ~**ARCHParser** ()  
*ARCHParser destructor.*
- std::vector< [BookParseEntry](#) > [arch\\_parser](#) (const std::filesystem::path &filepath)  
*This method carries out actual parsing.*
- void **stopAll** ()  
*Stops all operations.*

## Public Member Functions inherited from [LibArchive](#)

- [LibArchive](#) (const std::shared\_ptr< [AuxFunc](#) > &af)  
*[LibArchive](#) constructor.*
- std::filesystem::path [unpackByPosition](#) (const std::filesystem::path &archaddress, const std::filesystem::path &outfolder, const [ArchEntry](#) &entry)  
*Unpacks single entry content from zip archive.*
- std::string [unpackByPositionStr](#) (const std::filesystem::path &archaddress, const [ArchEntry](#) &entry)  
*Unpacks single entry content from zip archive.*
- std::filesystem::path [unpackByFileNameStream](#) (const std::filesystem::path &archaddress, const std::filesystem::path &outfolder, const std::string &filename)  
*Unpacks entry content from archive.*
- std::string [unpackByFileNameStreamStr](#) (const std::filesystem::path &archaddress, const std::string &filename)  
*Unpacks entry content from archive.*
- int [fileNames](#) (const std::filesystem::path &filepath, std::vector< [ArchEntry](#) > &filenames)  
*Lists all entries in archive file.*
- int [fileNamesStream](#) (const std::filesystem::path &address, std::vector< [ArchEntry](#) > &filenames)  
*Lists all entries in archive file.*
- [ArchEntry](#) [fileinfo](#) (const std::filesystem::path &address, const std::string &filename)  
*Returns [ArchEntry](#) for particular file or directory in archive.*
- int [libarchive\\_packing](#) (const std::filesystem::path &sourcepath, const std::filesystem::path &outpath)  
*Packs file or directory into archive.*
- int [libarchive\\_packing](#) (const std::shared\_ptr< archive > &a, const std::filesystem::path &sourcepath, const bool &rename\_source, const std::string &new\_source\_name)  
*Packs file or directory into archive.*
- [ArchiveRemoveEntry](#) [libarchive\\_remove\\_init](#) (const std::filesystem::path &sourcepath, const std::filesystem::path &outpath)  
*Initializes archive objects for removing entries from archive.*
- int [libarchive\\_remove\\_entry](#) ([ArchiveRemoveEntry](#) rm\_e, const std::vector< [ArchEntry](#) > &to\_remove)  
*Removes entry from archive.*
- void [libarchive\\_error](#) (const std::shared\_ptr< archive > &a, const std::string &message, const int &error\_number)  
*Prints libarchive error messages.*
- std::string [libarchive\\_read\\_entry\\_str](#) (archive \*a, archive\_entry \*entry)  
*Reads archived file to string.*
- int [libarchive\\_write\\_data](#) (archive \*a, const std::string &data)  
*Writes data to archive.*
- std::shared\_ptr< [ArchiveFileEntry](#) > [createArchFile](#) (const std::filesystem::path &archaddress, const int64\_t &position=la\_int64\_t(0))  
*Creates [ArchiveFileEntry](#) object.*
- std::shared\_ptr< archive > [libarchive\\_read\\_init](#) (std::shared\_ptr< [ArchiveFileEntry](#) > fl)  
*Initializes archive reading.*
- std::shared\_ptr< archive > [libarchive\\_read\\_init\\_fallback](#) (std::shared\_ptr< [ArchiveFileEntry](#) > fl)  
*Initializes archive reading.*
- std::filesystem::path [libarchive\\_read\\_entry](#) (archive \*a, archive\_entry \*entry, const std::filesystem::path &outfolder)  
*Unpacks libarchive entry content.*
- std::shared\_ptr< archive > [libarchive\\_write\\_init](#) (const std::filesystem::path &outpath)  
*Initializes writing to archive.*
- int [writeToArchive](#) (std::shared\_ptr< archive > a, const std::filesystem::path &source, const std::filesystem::path &path\_in\_arch)

*Writes file or directory to archive.*

- int [libarchive\\_write\\_directory](#) (archive \*a, archive\_entry \*entry, const std::filesystem::path &path\_in\_arch, const std::filesystem::path &source)

*Writes directory to archive.*

- int [libarchive\\_write\\_file](#) (archive \*a, archive\_entry \*entry, const std::filesystem::path &path\_in\_arch, const std::filesystem::path &source)

*Writes file to archive.*

- int [libarchive\\_write\\_data\\_from\\_file](#) (archive \*a, const std::filesystem::path &source)

*Writes raw data from file to archive.*

## 4.5.1 Detailed Description

The [ARCHParser](#) class.

Class for archives parsing. In most cases you do not need to use this class directly. Use [CreateCollection](#) or [RefreshCollection](#) instead.

## 4.5.2 Constructor & Destructor Documentation

### 4.5.2.1 ARCHParser()

```
ARCHParser::ARCHParser (
    const std::shared_ptr< AuxFunc > & af,
    const bool & rar_support)
```

[ARCHParser](#) constructor.

Parameters

<i>af</i>	smart pointer to <a href="#">AuxFunc</a> object.
<i>rar_support</i>	if <i>true</i> , <a href="#">ARCHParser</a> will parse rar archives, otherwise not.

## 4.5.3 Member Function Documentation

### 4.5.3.1 arch\_parser()

```
std::vector< BookParseEntry > ARCHParser::arch_parser (
    const std::filesystem::path & filepath)
```

This method carries out actual parsing.

Call this method to start archive parsing.

Parameters

<i>filepath</i>	absolute path to archive.
-----------------	---------------------------

Returns

Vector of [BookParseEntry](#) objects.

## 4.6 AuxFunc Class Reference

The [AuxFunc](#) class.

```
#include <AuxFunc.h>
```

### Public Member Functions

- virtual `~AuxFunc ()`  
*AuxFunc destructor.*
- `std::string to_utf_8 (const std::string &input, const char *conv_name)`  
*Converts string to UTF-8 string.*
- `std::string utf8_to_system (const std::string &input)`  
*Converts UTF-8 string to string in system default encoding.*
- `std::string utf_8_to (const std::string &input, const char *conv_name)`  
*Converts UTF-8 string to string in chosen encoding.*
- `const char * get_converter_by_number (const int32_t &num)`  
*Returns converter name.*
- `std::string detect_encoding (const std::string &buf)`  
*Tries to detect string encoding.*
- `std::filesystem::path homePath ()`  
*Returns user home directory path.*
- `std::filesystem::path get_selfpath ()`  
*Returns absolute path to program executable file.*
- `std::filesystem::path temp_path ()`  
*Returns absolute path to system temporary directory.*
- `std::filesystem::path share_path ()`  
*Returns absolute path to share directory, used by **MLBookProc**.*
- `std::vector< GenreGroup > get_genre_list ()`  
*Returns translated genre groups and genres names.*
- `std::string libgcrypt_error_handling (const gcry_error_t &err)`  
*Auxiliary method to reinterpret libgcrypt errors as strings.*
- `std::string to_hex (const std::string &source)`  
*Converts given string to hex format.*
- `std::string stringToLower (const std::string &line)`  
*Converts all letters of the string to lowercase letters.*
- `std::string randomFileName ()`  
*Returns random string.*
- `std::string time_t_to_date (const time_t &tt)`  
*Converts time\_t value to calendar date.*
- `bool if_supported_type (const std::filesystem::path &ch_p)`  
*Checks if given file is supported by **MLBookProc**.*
- `void html_to_utf8 (std::string &input)`  
*Convert 'html' symbols to UTF-8 characters.*
- `void open_book_callback (const std::filesystem::path &path)`  
*Opens given file in default system application.*
- `void copy_book_callback (const std::filesystem::path &source, const std::filesystem::path &out)`  
*Replaces out file by source file.*
- `std::vector< std::string > get_supported_types ()`  
*Returns supported file types.*

- `std::vector< std::string > get\_supported\_archive\_types\_packing ()`  
Same as [get\\_supported\\_types\(\)](#), but returns only archives types, available for packing.
- `std::vector< std::string > get\_supported\_archive\_types\_unpacking ()`  
Same as [get\\_supported\\_types\(\)](#), but returns only archives types, available for unpacking.
- `std::string get\_extension (const std::filesystem::path &p)`  
Returns file extension.
- `int32_t get\_charset\_conv\_quantity ()`  
Returns number of available converters.
- `bool get\_activated ()`  
Checks if dependencies have been successfully activated.
- `std::shared_ptr< ddjvu_context_t > getDJVUContext ()`  
Returns smart pointer to djvu context object.

### Static Public Member Functions

- `static std::shared_ptr< AuxFunc > create ()`  
Creates [AuxFunc](#) object.

## 4.6.1 Detailed Description

The [AuxFunc](#) class.

[AuxFunc](#) class contains various useful auxiliary methods. [AuxFunc](#) object must be created (see [create\(\)](#)) before using of any **MLBookProc** methods or classes. [create\(\)](#) method should be called only once per program. Also it is strongly recommended to call [get\\_activated\(\)](#) method immediately after [AuxFunc](#) object creation.

## 4.6.2 Member Function Documentation

### 4.6.2.1 [copy\\_book\\_callback\(\)](#)

```
void AuxFunc::copy_book_callback (
    const std::filesystem::path & source,
    const std::filesystem::path & out)
```

Replaces out file by source file.

This method acts like `std::filesystem::copy`. It was introduced due to MinGW bug (MinGW ignores `std::filesystem::copy_options::overwrite_existing`).

#### Parameters

<i>source</i>	file to be copied.
<i>out</i>	file to be replaced.

#### 4.6.2.2 create()

```
static std::shared_ptr< AuxFunc > AuxFunc::create () [static]
```

Creates [AuxFunc](#) object.

This method must be called before using of any **MLBookProc** classes or methods.

##### Warning

This method should be called only once per program.

##### Returns

Smart pointer to [AuxFunc](#) object.

#### 4.6.2.3 detect\_encoding()

```
std::string AuxFunc::detect_encoding (
    const std::string & buf)
```

Tries to detect string encoding.

##### Parameters

<i>buf</i>	string which encoding is to be detected.
------------	--

##### Returns

String containing encoding name.

#### 4.6.2.4 get\_activated()

```
bool AuxFunc::get_activated ()
```

Checks if dependencies have been successfully activated.

##### Returns

*true* if all dependencies have been successfully activated, *false* otherwise.

#### 4.6.2.5 get\_charset\_conv\_quantity()

```
int32_t AuxFunc::get_charset_conv_quantity ()
```

Returns number of available converters.

See [icu documentation](#) for details.

##### Returns

Signed 32-bit integer number of available convertors.

#### 4.6.2.6 get\_converter\_by\_number()

```
const char * AuxFunc::get_converter_by_number (
    const int32_t & num)
```

Returns converter name.

## Parameters

<i>num</i>	converter number (see <a href="#">icu documentation</a> for details).
------------	---

## Returns

Pointer to string, containing converter name.

#### 4.6.2.7 get\_extension()

```
std::string AuxFunc::get_extension (  
    const std::filesystem::path & p)
```

Returns file extension.

## Parameters

<i>p</i>	absolute path to file.
----------	------------------------

## Returns

File extension with beginning dot (looks like ".tar.gz").

#### 4.6.2.8 get\_genre\_list()

```
std::vector< GenreGroup > AuxFunc::get_genre_list ()
```

Returns translated genre groups and genres names.

Resulting genre groups and genres will be in default system language, if translations are available, or in English.

## Returns

Vector of [GenreGroup](#) objects.

#### 4.6.2.9 get\_selfpath()

```
std::filesystem::path AuxFunc::get_selfpath ()
```

Returns absolute path to program executable file.

## Returns

Absolute path to executable file.

#### 4.6.2.10 `get_supported_archive_types_packing()`

```
std::vector< std::string > AuxFunc::get_supported_archive_types_packing ()
```

Same as [get\\_supported\\_types\(\)](#), but returns only archives types, available for packing.

##### Returns

Vector of supported archives types.

#### 4.6.2.11 `get_supported_archive_types_unpacking()`

```
std::vector< std::string > AuxFunc::get_supported_archive_types_unpacking ()
```

Same as [get\\_supported\\_types\(\)](#), but returns only archives types, available for unpacking.

##### Returns

Vector of supported archives types.

#### 4.6.2.12 `get_supported_types()`

```
std::vector< std::string > AuxFunc::get_supported_types ()
```

Returns supported file types.

##### Returns

Vector of supported files extensions without beginning dot (looks like "fb2").

#### 4.6.2.13 `getDJVUContext()`

```
std::shared_ptr< ddjvu_context_t > AuxFunc::getDJVUContext ()
```

Returns smart pointer to djvu context object.

##### Warning

Resulting smart pointer can be *nullptr* (see [get\\_activated\(\)](#)).

##### Returns

Smart pointer to djvu context object.

#### 4.6.2.14 `homePath()`

```
std::filesystem::path AuxFunc::homePath ()
```

Returns user home directory path.

##### Returns

Absolute path to home directory.

#### 4.6.2.15 `html_to_utf8()`

```
void AuxFunc::html_to_utf8 (  
    std::string & input)
```

Converst 'html' symbols to UTF-8 characters.

Replaces "&#lt;unicode\_number>;" symbols by UTF-8 characters.



## Parameters

<i>input</i>	string to be converted.
--------------	-------------------------

**4.6.2.16 if\_supported\_type()**

```
bool AuxFunc::if_supported_type (
    const std::filesystem::path & ch_p)
```

Checks if given file is supported by **MLBookProc**.

'Supported types' check is carried out by file extension.

## Parameters

<i>ch_p</i>	absolute path to file.
-------------	------------------------

## Returns

*true* if file is supported, *false* otherwise.

**4.6.2.17 libgcrypt\_error\_handling()**

```
std::string AuxFunc::libgcrypt_error_handling (
    const gcry_error_t & err)
```

Auxiliary method to reinterpret libgcrypt errors as strings.

In most cases you do not need to call this method directly.

## Parameters

<i>err</i>	libgcrypt error code (see <a href="#">libgcrypt</a> documentation for details).
------------	---

## Returns

Human-readable string explaining error code.

**4.6.2.18 open\_book\_callback()**

```
void AuxFunc::open_book_callback (
    const std::filesystem::path & path)
```

Opens given file in default system application.

## Parameters

<i>path</i>	absolute path to file to be opened.
-------------	-------------------------------------

**4.6.2.19 randomFileName()**

```
std::string AuxFunc::randomFileName ()
```

Returns random string.

## Returns

Random string (it will look like "<random\_hex\_numbe>MLBookProc").

**4.6.2.20 share\_path()**

```
std::filesystem::path AuxFunc::share_path ()
```

Returns absolute path to *share* directory, used by **MLBookProc**.

Result path is calculating as path relative to program executable file path ('absolute\_path\_to\_executable\_↵ file/./share').

## Returns

Absolute path to *share* directory, used by **MLBookProc**.

**4.6.2.21 stringToLower()**

```
std::string AuxFunc::stringToLower (
    const std::string & line)
```

Converts all letters of the string to lowercase letters.

## Parameters

<i>line</i>	UTF-8 encoded string to be converted to lowercase.
-------------	--

## Returns

Lowercase string.

**4.6.2.22 temp\_path()**

```
std::filesystem::path AuxFunc::temp_path ()
```

Returns absolute path to system temporary directory.

## Returns

Absolute path to system temporary directory.

**4.6.2.23 time\_t\_to\_date()**

```
std::string AuxFunc::time_t_to_date (
    const time_t & tt)
```

Converts time\_t value to calendar date.

**Parameters**

<i>tt</i>	time_t value.
-----------	---------------

**Returns**

Date string (it will look like "<day\_number>.<month\_number>.<year>")

**4.6.2.24 to\_hex()**

```
std::string AuxFunc::to_hex (  
    const std::string & source)
```

Converts given string to hex format.

Each char element will be converted to two hexadecimal digits.

**Parameters**

<i>source</i>	string to be converted.
---------------	-------------------------

**Returns**

String in hex format.

**4.6.2.25 to\_utf\_8()**

```
std::string AuxFunc::to_utf_8 (  
    const std::string & input,  
    const char * conv_name)
```

Converts string to UTF-8 string.

**Parameters**

<i>input</i>	string to be converted.
<i>conv_name</i>	input string encoding name (see <a href="#">icu documentation</a> for details).

**Returns**

UTF-8 encoded string or empty string in case of any error.

**4.6.2.26 utf8\_to\_system()**

```
std::string AuxFunc::utf8_to_system (  
    const std::string & input)
```

Converts UTF-8 string to string in system default encoding.

## Parameters

<i>input</i>	UTF-8 string to be converted.
--------------	-------------------------------

## Returns

String in system encoding or empty string in case of any error.

## 4.6.2.27 utf\_8\_to()

```
std::string AuxFunc::utf_8_to (
    const std::string & input,
    const char * conv_name)
```

Converts UTF-8 string to string in chosen encoding.

## Parameters

<i>input</i>	string to be converted.
<i>conv_name</i>	output string encoding name (see <a href="#">icu documentation</a> for details).

## Returns

String in chosen encoding or empty string in case of any error.

## 4.7 BaseKeeper Class Reference

The [BaseKeeper](#) class.

```
#include <BaseKeeper.h>
```

## Public Member Functions

- [BaseKeeper](#) (const std::shared\_ptr< [AuxFunc](#) > &af)  
*BaseKeeper constructor.*
- virtual ~**BaseKeeper** ()  
*BaseKeeper destructor.*
- void [loadCollection](#) (const std::string &col\_name)  
*Loads collection database to memory.*
- std::vector< [BookBaseEntry](#) > [searchBook](#) (const [BookBaseEntry](#) &search)  
*Searches book in collection.*
- std::vector< std::string > [collectionAuthors](#) ()  
*Lists all authors, found in collection.*
- std::vector< [BookBaseEntry](#) > [booksWithNotes](#) (const std::vector< [NotesBaseEntry](#) > &notes)  
*Lists all books of current collection, which have notes.*
- void **stopSearch** ()  
*Stops all search operations.*
- void **clearBase** ()  
*Unloads collection base from memory.*
- std::vector< [FileParseEntry](#) > [get\\_base\\_vector](#) ()  
*Returns copy of inner database vector.*

## Static Public Member Functions

- static std::filesystem::path [get\\_books\\_path](#) (const std::string &collection\_name, const std::shared\_ptr<[AuxFunc](#) > &af)

*Returns absolute path to directory containing collection books.*

## Public Attributes

- std::function< void(const double &progr, const double &sz)> [auth\\_show\\_progr](#)  
*collectionAuthors() progress callback*

## 4.7.1 Detailed Description

The [BaseKeeper](#) class.

This class is intended to keep and operate collections databases. [loadCollection\(\)](#) method should be called first.

## 4.7.2 Constructor & Destructor Documentation

### 4.7.2.1 BaseKeeper()

```
BaseKeeper::BaseKeeper (
    const std::shared_ptr< AuxFunc > & af)
```

[BaseKeeper](#) constructor.

#### Parameters

<i>af</i>	smart pointer to <a href="#">AuxFunc</a> object.
-----------	--

## 4.7.3 Member Function Documentation

### 4.7.3.1 booksWithNotes()

```
std::vector< BookBaseEntry > BaseKeeper::booksWithNotes (
    const std::vector< NotesBaseEntry > & notes)
```

Lists all books of current collection, which have notes.

#### Parameters

<i>notes</i>	vector of notes (see <a href="#">NotesKeeper</a> class documentation).
--------------	--

#### Returns

Vector of books with notes.

#### 4.7.3.2 collectionAuthors()

```
std::vector< std::string > BaseKeeper::collectionAuthors ()
```

Lists all authors, found in collection.

##### Returns

Vector containing UTF-8 author's names strings.

#### 4.7.3.3 get\_base\_vector()

```
std::vector< FileParseEntry > BaseKeeper::get_base_vector ()
```

Returns copy of inner database vector.

##### Returns

Database vector.

#### 4.7.3.4 get\_books\_path()

```
static std::filesystem::path BaseKeeper::get_books_path (  
    const std::string & collection_name,  
    const std::shared_ptr< AuxFunc > & af) [static]
```

Returns absolute path to directory containing collection books.

This method can be called without collection loading to memory.

##### Note

This method can throw [MLException](#) in case of errors.

##### Parameters

<i>collection_name</i>	collection name.
<i>af</i>	smart pointer to <a href="#">AuxFunc</a> object.

##### Returns

Absolute path to books directory.

#### 4.7.3.5 loadCollection()

```
void BaseKeeper::loadCollection (  
    const std::string & col_name)
```

Loads collection database to memory.

##### Note

This method can throw [MLException](#) in case of errors.

## Parameters

<code>col_name</code>	collection name.
-----------------------	------------------

**4.7.3.6 searchBook()**

```
std::vector< BookBaseEntry > BaseKeeper::searchBook (
    const BookBaseEntry & search)
```

Searches book in collection.

[BookBaseEntry](#) object must be provided as search request. It is necessary to fill in any field in the inner [BookParseEntry](#) object to receive particular result. Otherwise complete collection book list will be returned.

## Parameters

<code>search</code>	<a href="#">BookBaseEntry</a> object.
---------------------	---------------------------------------

## Returns

Vector of [BookBaseEntry](#) objects, containing search results.

**4.7.4 Member Data Documentation****4.7.4.1 auth\_show\_progr**

```
std::function<void(const double &progr, const double &sz)> BaseKeeper::auth_show_progr
```

[collectionAuthors\(\)](#) progress callback

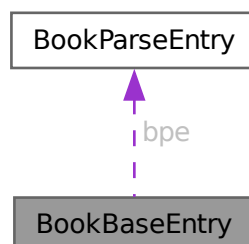
[collectionAuthors\(\)](#) method execution can take some time. This callback indicates progress. *progr* is current progress in conventional units. *sz* is total conventional units to be processed.

**4.8 BookBaseEntry Class Reference**

The [BookBaseEntry](#) class.

```
#include <BookBaseEntry.h>
```

Collaboration diagram for BookBaseEntry:



## Public Member Functions

- **BookBaseEntry** ()  
*BookBaseEntry* constructor.
- **BookBaseEntry** (const **BookBaseEntry** &other)  
*BookBaseEntry* copy constructor.
- **BookBaseEntry** (**BookBaseEntry** &&other)  
*BookBaseEntry* move constructor.
- **BookBaseEntry** & **operator=** (const **BookBaseEntry** &other)  
*operator =*
- **BookBaseEntry** & **operator=** (**BookBaseEntry** &&other)  
*operator =*
- bool **operator==** (const **BookBaseEntry** &other)  
*operator ==*
- **BookBaseEntry** (const **BookParseEntry** &bpe, const std::filesystem::path &book\_file\_path)  
*BookBaseEntry* constructor.

## Public Attributes

- std::filesystem::path **file\_path**  
*Absolute path to book file or archive.*
- **BookParseEntry** **bpe**  
*BookParseEntry* object.

## 4.8.1 Detailed Description

The **BookBaseEntry** class.

Auxiliary class, used to keep collections databases search requests and search results.

## 4.8.2 Constructor & Destructor Documentation

### 4.8.2.1 BookBaseEntry()

```
BookBaseEntry::BookBaseEntry (
    const BookParseEntry & bpe,
    const std::filesystem::path & book_file_path)
```

**BookBaseEntry** constructor.

#### Parameters

<i>bpe</i>	<b>BookParseEntry</b> object.
<i>book_file_path</i>	absolute path to books file or archive.



## 4.9 BookInfo Class Reference

The [BookInfo](#) class.

```
#include <BookInfo.h>
```

### Public Member Functions

- [BookInfo](#) (const std::shared\_ptr< [AuxFunc](#) > &af)  
*BookInfo constructor.*
- std::shared\_ptr< [BookInfoEntry](#) > [get\\_book\\_info](#) (const [BookBaseEntry](#) &bbe)  
*Retruns information about book.*
- void [set\\_dpi](#) (const double &h\_dpi, const double &v\_dpi)  
*Sets DPI.*

### 4.9.1 Detailed Description

The [BookInfo](#) class.

This class contains methods to get extra information (like annotation, cover, source paper book info, etc) from books.

### 4.9.2 Constructor & Destructor Documentation

#### 4.9.2.1 BookInfo()

```
BookInfo::BookInfo (
    const std::shared_ptr< AuxFunc > & af)
```

[BookInfo](#) constructor.

Parameters

<i>af</i>	smart pointer to <a href="#">AuxFunc</a> object.
-----------	--

### 4.9.3 Member Function Documentation

#### 4.9.3.1 get\_book\_info()

```
std::shared_ptr< BookInfoEntry > BookInfo::get_book_info (
    const BookBaseEntry & bbe)
```

Retruns information about book.

See also [set\\_dpi\(\)](#).

## Parameters

<i>bbe</i>	search result, returned by <a href="#">BaseKeeper::searchBook()</a> method.
------------	---

## Returns

Smart pointer to [BookInfoEntry](#) object containing various information about book.

**4.9.3.2 set\_dpi()**

```
void BookInfo::set_dpi (
    const double & h_dpi,
    const double & v_dpi)
```

Sets DPI.

This method should be called before [get\\_book\\_info\(\)](#). It sets **DPI** to display books cover correctly. Default values are 72.0 and 72.0. It is not compulsory to call this method, but it is highly recommended.

## Parameters

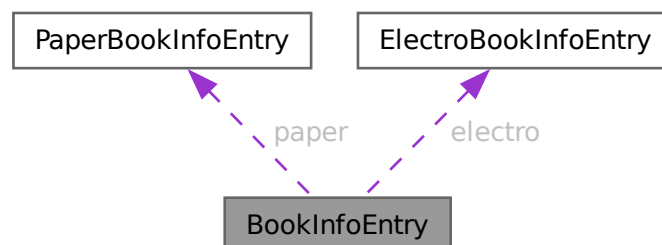
<i>h_dpi</i>	horizontal HREF=" <a href="https://en.wikipedia.org/wiki/Dots_per_inch">https://en.wikipedia.org/wiki/Dots_per_inch</a> ">DPI.
<i>v_dpi</i>	vertical HREF=" <a href="https://en.wikipedia.org/wiki/Dots_per_inch">https://en.wikipedia.org/wiki/Dots_per_inch</a> ">DPI.

**4.10 BookInfoEntry Class Reference**

The [BookInfoEntry](#) class.

```
#include <BookInfoEntry.h>
```

Collaboration diagram for BookInfoEntry:



## Public Types

- enum [cover\\_types](#) {  
[base64](#) , [file](#) , [rgb](#) , [rgba](#) ,  
[error](#) }  
*The cover types enumerator.*

## Public Member Functions

- **BookInfoEntry** ()  
[BookInfoEntry](#) constructor.
- virtual ~**BookInfoEntry** ()  
[BookInfoEntry](#) destructor.
- **BookInfoEntry** (const [BookInfoEntry](#) &other)  
[BookInfoEntry](#) copy constructor.
- **BookInfoEntry** ([BookInfoEntry](#) &&other)  
[BookInfoEntry](#) move constructor.
- [BookInfoEntry](#) & **operator=** (const [BookInfoEntry](#) &other)  
operator =
- [BookInfoEntry](#) & **operator=** ([BookInfoEntry](#) &&other)  
operator =

## Public Attributes

- std::string **annotation**  
*Book annotation.*
- std::string **cover**  
*Book cover image.*
- [cover\\_types](#) **cover\_type** = [cover\\_types::error](#)  
*Type of image.*
- std::string **language**  
*Book language.*
- std::string **src\_language**  
*Language of book source.*
- std::string **translator**  
*Translator.*
- [PaperBookInfoEntry](#) \* **paper** = nullptr  
*Pointer to [PaperBookInfoEntry](#).*
- [ElectroBookInfoEntry](#) \* **electro** = nullptr  
*Various technical information about book file. See [ElectroBookInfoEntry](#).*
- int **bytes\_per\_row** = 0  
*Number of bytes per row for RGB and RGBA images.*

### 4.10.1 Detailed Description

The [BookInfoEntry](#) class.

Auxiliary class keeping various extra information about book (see [BookInfo](#)).

### 4.10.2 Member Enumeration Documentation

#### 4.10.2.1 cover\_types

```
enum BookInfoEntry::cover\_types
```

The cover types enumerator.

## Enumerator

base64	base64 image
file	various image files formats, like JPG
rgb	RGB image
rgba	RGBA image
error	no image, default value

## 4.11 BookMarks Class Reference

The [BookMarks](#) class.

```
#include <BookMarks.h>
```

### Public Member Functions

- [BookMarks](#) (const std::shared\_ptr< [AuxFunc](#) > &af)  
*BookMarks constructor.*
- virtual ~**BookMarks** ()  
*BookMarks destructor.*
- int [createBookMark](#) (const std::string &col\_name, const [BookBaseEntry](#) &bbe)  
*Creates bookmark.*
- std::vector< std::tuple< std::string, [BookBaseEntry](#) > > [getBookMarks](#) ()  
*Returns bookmarks.*
- void [removeBookMark](#) (const std::string &col\_name, const [BookBaseEntry](#) &bbe)  
*Removes bookmark.*

### 4.11.1 Detailed Description

The [BookMarks](#) class.

This class keeps and operates collection bookmarks. Path to bookmarks base is "`~/local/share/MyLibrary/BookMarks/bookmarks`".

### 4.11.2 Constructor & Destructor Documentation

#### 4.11.2.1 BookMarks()

```
BookMarks::BookMarks (
    const std::shared_ptr< AuxFunc > & af)
```

[BookMarks](#) constructor.

#### Parameters

<i>af</i>	smart pointer to <a href="#">AuxFunc</a> object.
-----------	--

### 4.11.3 Member Function Documentation

#### 4.11.3.1 createBookMark()

```
int BookMarks::createBookMark (
    const std::string & col_name,
    const BookBaseEntry & bbe)
```

Creates bookmark.

If bookmark already exists, returns 0, in case of success returns 1, otherwise returns -1.

##### Parameters

<i>col_name</i>	collection name book came from.
<i>bbe</i>	<a href="#">BookBaseEntry</a> got from <a href="#">BaseKeeper::searchBook()</a> .

##### Returns

Error code.

#### 4.11.3.2 getBookMarks()

```
std::vector< std::tuple< std::string, BookBaseEntry > > BookMarks::getBookMarks ()
```

Returns bookmarks.

##### Returns

Vector of bookmarks tuples. First element of tuple is collection name, book came from. Second element is book entry.

#### 4.11.3.3 removeBookMark()

```
void BookMarks::removeBookMark (
    const std::string & col_name,
    const BookBaseEntry & bbe)
```

Removes bookmark.

##### Parameters

<i>col_name</i>	collection name.
<i>bbe</i>	book entry to be removed.

## 4.12 BookParseEntry Class Reference

The [BookParseEntry](#) class.

```
#include <BookParseEntry.h>
```

## Public Member Functions

- **BookParseEntry** ()  
*BookParseEntry* constructor.
- **BookParseEntry** (const **BookParseEntry** &other)  
*BookParseEntry* copy constructor.
- **BookParseEntry** (**BookParseEntry** &&other)  
*BookParseEntry* move constructor.
- **BookParseEntry** & **operator=** (const **BookParseEntry** &other)  
*operator =*
- **BookParseEntry** & **operator=** (**BookParseEntry** &&other)  
*operator =*
- bool **operator==** (const **BookParseEntry** &other)  
*operator ==*

## Public Attributes

- std::string **book\_path**  
*Path to book in file (in case of archive, empty otherwise).*
- std::string **book\_author**  
*Book author (if any, empty otherwise).*
- std::string **book\_name**  
*Book name.*
- std::string **book\_series**  
*Book series (if any, empty otherwise).*
- std::string **book\_genre**  
*Book genres(s) (if any, empty otherwise).*
- std::string **book\_date**  
*Book creation date (if available in metadata, empty otherwise).*

### 4.12.1 Detailed Description

The **BookParseEntry** class.

Auxiliary class keeping relative path to book in file (in case of archive, empty otherwise), book author(s), book name, book series, book genre(s), date of book creation (if available).

### 4.12.2 Member Data Documentation

#### 4.12.2.1 book\_genre

```
std::string BookParseEntry::book_genre
```

Book genres(s) (if any, empty otherwise).

List of genres separated by ", " sequence.

#### 4.12.2.2 book\_name

```
std::string BookParseEntry::book_name
```

Book name.

Must not be empty. If book name cannot be obtained from book metadata, book file name will be used.

#### 4.12.2.3 book\_path

```
std::string BookParseEntry::book_path
```

Path to book in file (in case of archive, empty otherwise).

In case of "archive inside archive" situation "\n" (ASCII new line) symbol used as separator. It means that path to book inside archive looks like "<archive\_one>\n<archive\_two>\n<archive\_three>\n<book\_file>" or "<archive\_one>\n<book\_file>".

## 4.13 ByteOrder Class Reference

The [ByteOrder](#) class.

```
#include <ByteOrder.h>
```

### Public Member Functions

- **ByteOrder** ()  
*ByteOrder constructor.*
- virtual ~**ByteOrder** ()  
*ByteOrder destructor.*
- **ByteOrder** (const [ByteOrder](#) &other)  
*ByteOrder copy constructor.*
- [ByteOrder](#) (uint64\_t val)  
*ByteOrder constructor.*
- [ByteOrder](#) (uint32\_t val)  
*ByteOrder constructor.*
- [ByteOrder](#) (uint16\_t val)  
*ByteOrder constructor.*
- [ByteOrder](#) (int64\_t val)  
*ByteOrder constructor.*
- [ByteOrder](#) (int32\_t val)  
*ByteOrder constructor.*
- [ByteOrder](#) (int16\_t val)  
*ByteOrder constructor.*
- [ByteOrder](#) (float val)  
*ByteOrder constructor.*
- [ByteOrder](#) (double val)  
*ByteOrder constructor.*
- [ByteOrder](#) & **operator=** (const [ByteOrder](#) &other)

- operator =*
- **ByteOrder** & **operator=** (const uint64\_t &val)
- operator =*
- **ByteOrder** & **operator=** (const uint32\_t &val)
- operator =*
- **ByteOrder** & **operator=** (const uint16\_t &val)
- operator =*
- **ByteOrder** & **operator=** (const int64\_t &val)
- operator =*
- **ByteOrder** & **operator=** (const int32\_t &val)
- operator =*
- **ByteOrder** & **operator=** (const int16\_t &val)
- operator =*
- **ByteOrder** & **operator=** (const float &val)
- operator =*
- **ByteOrder** & **operator=** (const double &val)
- operator =*
- **operator uint64\_t** ()
- Returns number in native byte order.*
- **operator uint32\_t** ()
- Returns number in native byte order.*
- **operator uint16\_t** ()
- Returns number in native byte order.*
- **operator int64\_t** ()
- Returns number in native byte order.*
- **operator int32\_t** ()
- Returns number in native byte order.*
- **operator int16\_t** ()
- Returns number in native byte order.*
- **operator float** ()
- Returns number in native byte order.*
- **operator double** ()
- Returns number in native byte order.*
- template<typename T>  
void **get\_native** (T &result)
- Returns number in native byte order.*
- template<typename T>  
void **get\_big** (T &result)
- Returns "big endian" number.*
- template<typename T>  
void **get\_little** (T &result)
- Returns "little endian" number.*
- template<typename T>  
void **set\_big** (T val)
- Sets inner value to val.*
- template<typename T>  
void **set\_little** (T val)
- Sets inner value to val.*



### 4.13.1 Detailed Description

The [ByteOrder](#) class.

Auxiliary class. Is used to convert numbers to different byte orders.

#### Warning

If you get number from [ByteOrder](#), resulting number must have same type as input value, otherwise behavior is undefined.

### 4.13.2 Constructor & Destructor Documentation

#### 4.13.2.1 [ByteOrder\(\)](#) [1/8]

```
ByteOrder::ByteOrder (  
    uint64_t val)
```

[ByteOrder](#) constructor.

##### Parameters

<i>val</i>	value in native byte order.
------------	-----------------------------

#### 4.13.2.2 [ByteOrder\(\)](#) [2/8]

```
ByteOrder::ByteOrder (  
    uint32_t val)
```

[ByteOrder](#) constructor.

##### Parameters

<i>val</i>	value in native byte order.
------------	-----------------------------

#### 4.13.2.3 [ByteOrder\(\)](#) [3/8]

```
ByteOrder::ByteOrder (  
    uint16_t val)
```

[ByteOrder](#) constructor.

##### Parameters

<i>val</i>	value in native byte order.
------------	-----------------------------

#### 4.13.2.4 [ByteOrder\(\)](#) [4/8]

```
ByteOrder::ByteOrder (  
    int64_t val)
```

[ByteOrder](#) constructor.

## Parameters

<i>val</i>	value in native byte order.
------------	-----------------------------

**4.13.2.5 ByteOrder()** [5/8]

```
ByteOrder::ByteOrder (  
    int32_t val)
```

[ByteOrder](#) constructor.

## Parameters

<i>val</i>	value in native byte order.
------------	-----------------------------

**4.13.2.6 ByteOrder()** [6/8]

```
ByteOrder::ByteOrder (  
    int16_t val)
```

[ByteOrder](#) constructor.

## Parameters

<i>val</i>	value in native byte order.
------------	-----------------------------

**4.13.2.7 ByteOrder()** [7/8]

```
ByteOrder::ByteOrder (  
    float val)
```

[ByteOrder](#) constructor.

## Parameters

<i>val</i>	value in native byte order.
------------	-----------------------------

**4.13.2.8 ByteOrder()** [8/8]

```
ByteOrder::ByteOrder (  
    double val)
```

[ByteOrder](#) constructor.

## Parameters

<i>val</i>	value in native byte order.
------------	-----------------------------

### 4.13.3 Member Function Documentation

#### 4.13.3.1 get\_big()

```
template<typename T>
void ByteOrder::get_big (
    T & result)
```

Returns "big endian" number.

## Parameters

<i>result</i>	resulting number.
---------------	-------------------

#### 4.13.3.2 get\_little()

```
template<typename T>
void ByteOrder::get_little (
    T & result)
```

Returns "little endian" number.

## Parameters

<i>result</i>	resulting number.
---------------	-------------------

#### 4.13.3.3 get\_native()

```
template<typename T>
void ByteOrder::get_native (
    T & result)
```

Returns number in native byte order.

## Parameters

<i>result</i>	resulting number.
---------------	-------------------

#### 4.13.3.4 operator=() [1/8]

```
ByteOrder & ByteOrder::operator= (
    const double & val)
```

operator =

## Parameters

<i>val</i>	value in native byte order.
------------	-----------------------------

**4.13.3.5 operator=()** [2/8]

```
ByteOrder & ByteOrder::operator= (  
    const float & val)
```

operator =

## Parameters

<i>val</i>	value in native byte order.
------------	-----------------------------

**4.13.3.6 operator=()** [3/8]

```
ByteOrder & ByteOrder::operator= (  
    const int16_t & val)
```

operator =

## Parameters

<i>val</i>	value in native byte order.
------------	-----------------------------

**4.13.3.7 operator=()** [4/8]

```
ByteOrder & ByteOrder::operator= (  
    const int32_t & val)
```

operator =

## Parameters

<i>val</i>	value in native byte order.
------------	-----------------------------

**4.13.3.8 operator=()** [5/8]

```
ByteOrder & ByteOrder::operator= (  
    const int64_t & val)
```

operator =

## Parameters

<i>val</i>	value in native byte order.
------------	-----------------------------

**4.13.3.9 operator=()** [6/8]

```
ByteOrder & ByteOrder::operator= (
    const uint16_t & val)
```

operator =

## Parameters

<i>val</i>	value in native byte order.
------------	-----------------------------

**4.13.3.10 operator=()** [7/8]

```
ByteOrder & ByteOrder::operator= (
    const uint32_t & val)
```

operator =

## Parameters

<i>val</i>	value in native byte order.
------------	-----------------------------

**4.13.3.11 operator=()** [8/8]

```
ByteOrder & ByteOrder::operator= (
    const uint64_t & val)
```

operator =

## Parameters

<i>val</i>	value in native byte order.
------------	-----------------------------

**4.13.3.12 set\_big()**

```
template<typename T>
void ByteOrder::set_big (
    T val)
```

Sets inner value to val.

Sets inner value to val. val will be processed as "big endian".

**Parameters**

<i>val</i>	value to be set.
------------	------------------

**4.13.3.13 set\_little()**

```
template<typename T>
void ByteOrder::set_little (
    T val)
```

Sets inner value to val.

Sets inner value to val. val will be processed as "little endian".

**Parameters**

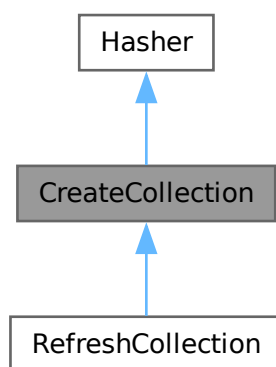
<i>val</i>	value to be set.
------------	------------------

**4.14 CreateCollection Class Reference**

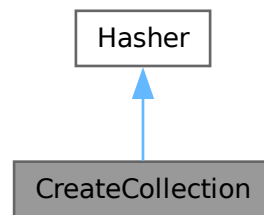
The [CreateCollection](#) class.

```
#include <CreateCollection.h>
```

Inheritance diagram for CreateCollection:



Collaboration diagram for CreateCollection:



### Public Member Functions

- `CreateCollection` (const std::shared\_ptr< [AuxFunc](#) > &af, const std::filesystem::path &collection\_path, const std::filesystem::path &books\_path, const bool &rar\_support, const int &num\_threads)  
*CreateCollection constructor.*
- virtual ~`CreateCollection` ()  
*CreateCollection destructor.*
- void `createCollection` ()  
*Starts collection creation.*

### Public Member Functions inherited from [Hasher](#)

- `Hasher` (const std::shared\_ptr< [AuxFunc](#) > &af)  
*Hasher constructor.*
- std::string `buf_hashing` (const std::string &buf)  
*Creates hash sum for given buffer.*
- std::string `file_hashing` (const std::filesystem::path &filepath)  
*Creates hash sum for given file.*
- void `cancelAll` ()  
*Stops all operations.*

### Public Attributes

- std::function< void()> `pulse`  
*"Pulse" callback.*
- std::function< void(const double &)> `signal_total_bytes`  
*"Total bytes" callback.*
- std::function< void(const double &progress)> `progress`  
*"Progress" callback.*

### Protected Member Functions

- [CreateCollection](#) (const std::shared\_ptr< [AuxFunc](#) > &af, const int &num\_threads)  
*CreateCollection constructor.*
- void [threadRegulator](#) ()  
*Threads regulator.*
- void [openBaseFile](#) ()  
*Opens database file for writing.*
- void [closeBaseFile](#) ()  
*Finishes database writing.*
- void [write\\_file\\_to\\_base](#) (const [FileParseEntry](#) &fe)  
*Writes file data to database.*

### Protected Attributes

- std::filesystem::path [base\\_path](#)  
*Absolute path to database.*
- std::filesystem::path [books\\_path](#)  
*Absolute path to books directory.*
- bool [rar\\_support](#) = false  
*If true, rar archives will be processed, otherwise - not.*
- std::vector< std::tuple< std::filesystem::path, std::string > > [already\\_hashed](#)  
*Hashed files.*
- std::vector< std::filesystem::path > [need\\_to\\_parse](#)  
*"Need to parse" vector.*
- std::atomic< double > [current\\_bytes](#)  
*Keeps quantity of bytes have been processed.*

### Protected Attributes inherited from [Hasher](#)

- std::atomic< bool > [cancel](#)  
*Stops all operations if true.*
- std::function< void()> [stop\\_all\\_signal](#)  
*Stop signal for heir classes.*

#### 4.14.1 Detailed Description

The [CreateCollection](#) class.

This class contains methods for collection database creation.

#### 4.14.2 Constructor & Destructor Documentation

##### 4.14.2.1 [CreateCollection](#)() [1/2]

```
CreateCollection::CreateCollection (
    const std::shared_ptr< AuxFunc > & af,
    const std::filesystem::path & collection_path,
    const std::filesystem::path & books_path,
    const bool & rar_support,
    const int & num_threads)
```

[CreateCollection](#) constructor.



## Parameters

<i>af</i>	smart pointer to <a href="#">AuxFunc</a> object.
<i>collection_path</i>	absolute path to collection base directory (if it doesnt exist, <b>MLBookProc</b> will create it).
<i>books_path</i>	absolute path to books directory.
<i>rar_support</i>	if <i>true</i> , rar archives will be processed, otherwise - not (some rar archives can cause errors).
<i>num_threads</i>	limit of working threads to be used.

## 4.14.2.2 CreateCollection() [2/2]

```
CreateCollection::CreateCollection (
    const std::shared_ptr< AuxFunc > & af,
    const int & num_threads) [protected]
```

[CreateCollection](#) constructor.

## Warning

Do not call this constructor yourself!

This constructor is used by [RefreshCollection](#) class.

## Parameters

<i>af</i>	smart pointer to <a href="#">AuxFunc</a> object.
<i>num_threads</i>	limit of working threads to be used.

## 4.14.3 Member Function Documentation

## 4.14.3.1 closeBaseFile()

```
void CreateCollection::closeBaseFile () [protected]
```

Finishes database writing.

## Warning

Do not call this method yourself!

## 4.14.3.2 createCollection()

```
void CreateCollection::createCollection ()
```

Starts collection creation.

## Note

This method can throw [MLException](#) in case of errors.

#### 4.14.3.3 openBaseFile()

```
void CreateCollection::openBaseFile () [protected]
```

Opens database file for writing.

##### Warning

Do not call this method yourself!

#### 4.14.3.4 threadRegulator()

```
void CreateCollection::threadRegulator () [protected]
```

Threads regulator.

##### Warning

Do not call this method yourself!

#### 4.14.3.5 write\_file\_to\_base()

```
void CreateCollection::write_file_to_base (  
    const FileParseEntry & fe) [protected]
```

Writes file data to database.

##### Warning

Do not call this method yourself!

##### Parameters

<i>fe</i>	<a href="#">FileParseEntry</a> object.
-----------	--

### 4.14.4 Member Data Documentation

#### 4.14.4.1 already\_hashed

```
std::vector<std::tuple<std::filesystem::path, std::string> > CreateCollection::already_hashed  
[protected]
```

Hashed files.

This vector is used by [RefreshCollection](#) class to indicate files, has been already hashed.

##### Warning

Do not call or set this vector yourself!

#### 4.14.4.2 base\_path

```
std::filesystem::path CreateCollection::base_path [protected]
```

Absolute path to database.

##### Warning

Do not call or set this variable yourself!

#### 4.14.4.3 books\_path

```
std::filesystem::path CreateCollection::books_path [protected]
```

Absolute path to books directory.

##### Warning

Do not call or set this variable yourself!

#### 4.14.4.4 current\_bytes

```
std::atomic<double> CreateCollection::current_bytes [protected]
```

Keeps quantity of bytes have been processed.

##### Warning

Do not call or set this variable yourself!

#### 4.14.4.5 need\_to\_parse

```
std::vector<std::filesystem::path> CreateCollection::need_to_parse [protected]
```

"Need to parse" vector.

This vector is used to indicate files, needed to be parsed.

##### Warning

Do not call or set this vector yourself!

#### 4.14.4.6 progress

```
std::function<void(const double &progress)> CreateCollection::progress
```

"Progress" callback.

Emitted after each file processing completion. Indicates total quantity of bytes has been processed. Bind your method to it, if you need such information.

#### 4.14.4.7 pulse

```
std::function<void()> CreateCollection::pulse
```

"Pulse" callback.

Emitted while preliminary files collecting to show that process is not frozen. Bind your method to it, if you need such information.

#### 4.14.4.8 rar\_support

```
bool CreateCollection::rar_support = false [protected]
```

If *true*, rar archives will be processed, otherwise - not.

##### Warning

Do not call or set this variable yourself!

#### 4.14.4.9 signal\_total\_bytes

```
std::function<void(const double &)> CreateCollection::signal_total_bytes
```

"Total bytes" callback.

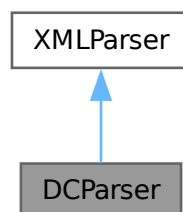
Emitted after preliminary files collecting completed to indicate total quantity of bytes to be processed. Bind your method to it, if you need such information.

## 4.15 DCParser Class Reference

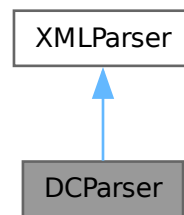
The [DCParser](#) class.

```
#include <DCParser.h>
```

Inheritance diagram for DCParser:



Collaboration diagram for DCParse:



### Public Member Functions

- `DCParse` (const std::shared\_ptr< [AuxFunc](#) > &af)  
*DCParse constructor.*
- std::string `dcTitle` (const std::string &dc\_file\_content, const std::vector< [XMLTag](#) > &tgvs)  
*Gets book title.*
- std::string `dcAuthor` (const std::string &dc\_file\_content, const std::vector< [XMLTag](#) > &tgvs)  
*Gets book author.*
- std::string `dcGenre` (const std::string &dc\_file\_content, const std::vector< [XMLTag](#) > &tgvs)  
*Gets book genre.*
- std::string `dcDate` (const std::string &dc\_file\_content, const std::vector< [XMLTag](#) > &tgvs)  
*Gets book creation date.*
- std::string `dcLanguage` (const std::string &dc\_file\_content, const std::vector< [XMLTag](#) > &tgvs)  
*Gets book language.*
- std::string `dcPublisher` (const std::string &dc\_file\_content, const std::vector< [XMLTag](#) > &tgvs)  
*Gets file publisher.*
- std::string `dcIdentifier` (const std::string &dc\_file\_content, const std::vector< [XMLTag](#) > &tgvs)  
*Gets book identifier.*
- std::string `dcSource` (const std::string &dc\_file\_content, const std::vector< [XMLTag](#) > &tgvs)  
*Gets book source.*
- std::string `dcDescription` (const std::string &dc\_file\_content, const std::vector< [XMLTag](#) > &tgvs)  
*Gets book description.*

### Public Member Functions inherited from [XMLParser](#)

- `XMLParser` (const std::shared\_ptr< [AuxFunc](#) > &af)  
*XMLParser constructor.*
- std::vector< [XMLTag](#) > `get_tag` (const std::string &book, const std::string &tag\_id)  
*Returns all tags with particular name.*
- std::string `get_book_encoding` (const std::string &book)  
*Returns [XML](#) document encoding.*
- std::string `get_element_attribute` (const std::string &element, const std::string &attr\_name)  
*Returns [XML](#) tag attribute if it was found.*
- std::vector< [XMLTag](#) > `listAllTags` (const std::string &book)

- Parses XML document.*
- void [searchTag](#) (const std::vector< [XMLTag](#) > &list, const std::string &tag\_id, std::vector< [XMLTag](#) > &result)  
*Searches tag in tag list.*
- void [htmlSymbolsReplacement](#) (std::string &book)  
*Replaces symbols encoded by "&..." sequences.*
- void [removeAllTags](#) (std::string &book)  
*Removes all tag elements from XML document.*

### 4.15.1 Detailed Description

The [DCParser](#) class.

Auxiliary class. Contains methods for [DublinCore](#) files parsing. This class is used in [ODTParser](#) and [EPUBParser](#). You do not need to call this class methods directly.

### 4.15.2 Constructor & Destructor Documentation

#### 4.15.2.1 DCParser()

```
DCParser::DCParser (
    const std::shared_ptr< AuxFunc > & af)
```

[DCParser](#) constructor.

Parameters

<i>af</i>	smart pointer to <a href="#">AuxFunc</a> object.
-----------	--

### 4.15.3 Member Function Documentation

#### 4.15.3.1 dcAuthor()

```
std::string DCParser::dcAuthor (
    const std::string & dc_file_content,
    const std::vector< XMLTag > & tgv)
```

Gets book author.

This method can be used to get author(s) from [DublinCore](#) file.

Parameters

<i>dc_file_content</i>	string containing file content.
<i>tgv</i>	<a href="#">XMLTag</a> vector obtained from <a href="#">XMLParser::listAllTags</a> method.

Returns

Author if any, empty string otherwise.

#### 4.15.3.2 dcDate()

```
std::string DCParser::dcDate (
    const std::string & dc_file_content,
    const std::vector< XMLTag > & tgv)
```

Gets book creation date.

This method can be used to get creation date from [DublinCore](#) file.

## Parameters

<i>dc_file_content</i>	string containing file content.
<i>tgvs</i>	<a href="#">XMLTag</a> vector obtained from <a href="#">XMLParser::listAllTags</a> method.

## Returns

File creation date if any, empty otherwise.

**4.15.3.3 dcDescription()**

```
std::string DCParse::dcDescription (
    const std::string & dc_file_content,
    const std::vector< XMLTag > & tgvs)
```

Gets book description.

This method can be used to get book description from [DublinCore](#) file.

## Parameters

<i>dc_file_content</i>	string containing file content.
<i>tgvs</i>	<a href="#">XMLTag</a> vector obtained from <a href="#">XMLParser::listAllTags</a> method.

## Returns

Book description if any, empty otherwise.

**4.15.3.4 dcGenre()**

```
std::string DCParse::dcGenre (
    const std::string & dc_file_content,
    const std::vector< XMLTag > & tgvs)
```

Gets book genre.

This method can be used to get genre(s) from [DublinCore](#) file.

## Parameters

<i>dc_file_content</i>	string containing file content.
<i>tgvs</i>	<a href="#">XMLTag</a> vector obtained from <a href="#">XMLParser::listAllTags</a> method.

## Returns

Genre if any, empty string otherwise.

**4.15.3.5 dcIdentifier()**

```
std::string DCParse::dcIdentifier (
    const std::string & dc_file_content,
    const std::vector< XMLTag > & tgvs)
```

Gets book identifier.

This method can be used to get identifier from [DublinCore](#) file.

## Parameters

<i>dc_file_content</i>	string containing file content.
<i>tgvs</i>	<a href="#">XMLTag</a> vector obtained from <a href="#">XMLParser::listAllTags</a> method.

## Returns

Book identifier if any, empty otherwise.

**4.15.3.6 dcLanguage()**

```
std::string DCParser::dcLanguage (
    const std::string & dc_file_content,
    const std::vector< XMLTag > & tgvs)
```

Gets book language.

This method can be used to get language from [DublinCore](#) file.

## Parameters

<i>dc_file_content</i>	string containing file content.
<i>tgvs</i>	<a href="#">XMLTag</a> vector obtained from <a href="#">XMLParser::listAllTags</a> method.

## Returns

File language if set, empty otherwise.

**4.15.3.7 dcPublisher()**

```
std::string DCParser::dcPublisher (
    const std::string & dc_file_content,
    const std::vector< XMLTag > & tgvs)
```

Gets file publisher.

This method can be used to get publisher from [DublinCore](#) file.

## Parameters

<i>dc_file_content</i>	string containing file content.
<i>tgvs</i>	<a href="#">XMLTag</a> vector obtained from <a href="#">XMLParser::listAllTags</a> method.

## Returns

File publisher if set, empty otherwise.

**4.15.3.8 dcSource()**

```
std::string DCParser::dcSource (
    const std::string & dc_file_content,
    const std::vector< XMLTag > & tgvs)
```

Gets book source.

This method can be used to get book source from [DublinCore](#) file.



## Parameters

<i>dc_file_content</i>	string containing file content.
<i>tgvl</i>	<a href="#">XMLTag</a> vector obtained from <a href="#">XMLParser::listAllTags</a> method.

## Returns

Book source if set, empty otherwise.

**4.15.3.9 dcTitle()**

```
std::string DCParser::dcTitle (
    const std::string & dc_file_content,
    const std::vector< XMLTag > & tgvl)
```

Gets book title.

This method can be used to get title from [DublinCore](#) file.

## Parameters

<i>dc_file_content</i>	string containing file content.
<i>tgvl</i>	<a href="#">XMLTag</a> vector obtained from <a href="#">XMLParser::listAllTags</a> method.

## Returns

Title if any, empty string otherwise.

**4.16 DJVUParser Class Reference**

The [DJVUParser](#) class.

```
#include <DJVUParser.h>
```

**Public Member Functions**

- [DJVUParser](#) (const std::shared\_ptr< [AuxFunc](#) > &af)  
*DJVUParser constructor.*
- [BookParseEntry djvu\\_parser](#) (const std::filesystem::path &filepath)  
*Parses djvu book.*
- std::shared\_ptr< [BookInfoEntry](#) > [djvu\\_book\\_info](#) (const std::filesystem::path &filepath)  
*Returns book info and book cover.*

**4.16.1 Detailed Description**

The [DJVUParser](#) class.

This class contains various methods for djvu books processing. In most cases you do not need to use this class directly. Use [CreateCollection](#), [RefreshCollection](#) and [BookInfo](#) instead.

## 4.16.2 Constructor & Destructor Documentation

### 4.16.2.1 DJVUParser()

```
DJVUParser::DJVUParser (  
    const std::shared_ptr< AuxFunc > & af)
```

[DJVUParser](#) constructor.

#### Parameters

<i>af</i>	smart pointer to <a href="#">AuxFunc</a> object.
-----------	--

## 4.16.3 Member Function Documentation

### 4.16.3.1 djvu\_book\_info()

```
std::shared_ptr< BookInfoEntry > DJVUParser::djvu_book_info (  
    const std::filesystem::path & filepath)
```

Returns book info and book cover.

#### Parameters

<i>filepath</i>	absolute path to djvu book.
-----------------	-----------------------------

#### Returns

Smart pointer to [BookInfoEntry](#) object.

### 4.16.3.2 djvu\_parser()

```
BookParseEntry DJVUParser::djvu_parser (  
    const std::filesystem::path & filepath)
```

Parses djvu book.

#### Parameters

<i>filepath</i>	absolute path to djvu book.
-----------------	-----------------------------

#### Returns

[BookParseEntry](#) object.

## 4.17 ElectroBookInfoEntry Class Reference

The [ElectroBookInfoEntry](#) class.

```
#include <ElectroBookInfoEntry.h>
```

### Public Member Functions

- **ElectroBookInfoEntry** ()  
*ElectroBookInfoEntry* constructor.
- **ElectroBookInfoEntry** (const [ElectroBookInfoEntry](#) &other)  
*ElectroBookInfoEntry* copy constructor.
- **ElectroBookInfoEntry** ([ElectroBookInfoEntry](#) &&other)  
*ElectroBookInfoEntry* move constructor.
- **ElectroBookInfoEntry** & **operator=** (const [ElectroBookInfoEntry](#) &other)  
*operator =*
- **ElectroBookInfoEntry** & **operator=** ([ElectroBookInfoEntry](#) &&other)  
*operator =*

### Public Attributes

- bool **available** = false  
*Indicates if any information is available.*
- std::string **author**  
*Author of file.*
- std::string **program\_used**  
*Program used to create file.*
- std::string **date**  
*Date of file creation.*
- std::string **src\_url**  
*Source URL if this document is a conversion of some other (online) document.*
- std::string **src\_ocr**  
*Author of the original (online) document, if this is a conversion.*
- std::string **id**  
*Unique file identifier.*
- std::string **version**  
*File version.*
- std::string **history**  
*History of file changes.*
- std::string **publisher**  
*File publisher.*

### 4.17.1 Detailed Description

The [ElectroBookInfoEntry](#) class.

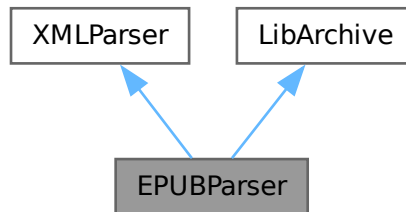
Auxiliary class containing various technical info about digital book file.

## 4.18 EPUBParser Class Reference

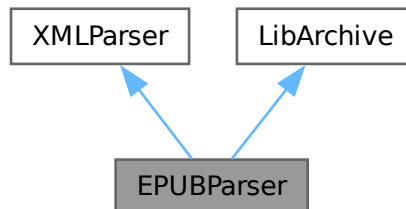
The [EPUBParser](#) class.

```
#include <EPUBParser.h>
```

Inheritance diagram for EPUBParser:



Collaboration diagram for EPUBParser:



### Public Member Functions

- [EPUBParser](#) (const std::shared\_ptr< [AuxFunc](#) > &af)  
*EPUBParser constructor.*
- virtual ~[EPUBParser](#) ()  
*EPUBParser destructor.*
- [BookParseEntry epub\\_parser](#) (const std::filesystem::path &filepath)  
*Parses epub book.*
- std::shared\_ptr< [BookInfoEntry](#) > [epub\\_book\\_info](#) (const std::filesystem::path &filepath)  
*Returns epub book info and cover.*

## Public Member Functions inherited from XMLParser

- [XMLParser](#) (const std::shared\_ptr< [AuxFunc](#) > &af)  
*XMLParser constructor.*
- std::vector< [XMLTag](#) > [get\\_tag](#) (const std::string &book, const std::string &tag\_id)  
*Returns all tags with particular name.*
- std::string [get\\_book\\_encoding](#) (const std::string &book)  
*Returns [XML](#) document encoding.*
- std::string [get\\_element\\_attribute](#) (const std::string &element, const std::string &attr\_name)  
*Returns [XML](#) tag attribute if it was found.*
- std::vector< [XMLTag](#) > [listAllTags](#) (const std::string &book)  
*Parses [XML](#) document.*
- void [searchTag](#) (const std::vector< [XMLTag](#) > &list, const std::string &tag\_id, std::vector< [XMLTag](#) > &result)  
*Searches tag in tag list.*
- void [htmlSymbolsReplacement](#) (std::string &book)  
*Replaces symbols encoded by "&..." sequences.*
- void [removeAllTags](#) (std::string &book)  
*Removes all tag elements from [XML](#) document.*

## Public Member Functions inherited from LibArchive

- [LibArchive](#) (const std::shared\_ptr< [AuxFunc](#) > &af)  
*LibArchive constructor.*
- std::filesystem::path [unpackByPosition](#) (const std::filesystem::path &archaddress, const std::filesystem::path &outfolder, const [ArchEntry](#) &entry)  
*Unpacks single entry content from zip archive.*
- std::string [unpackByPositionStr](#) (const std::filesystem::path &archaddress, const [ArchEntry](#) &entry)  
*Unpacks single entry content from zip archive.*
- std::filesystem::path [unpackByFileNameStream](#) (const std::filesystem::path &archaddress, const std::filesystem::path &outfolder, const std::string &filename)  
*Unpacks entry content from archive.*
- std::string [unpackByFileNameStreamStr](#) (const std::filesystem::path &archaddress, const std::string &filename)  
*Unpacks entry content from archive.*
- int [fileNames](#) (const std::filesystem::path &filepath, std::vector< [ArchEntry](#) > &filenames)  
*Lists all entries in archive file.*
- int [fileNamesStream](#) (const std::filesystem::path &address, std::vector< [ArchEntry](#) > &filenames)  
*Lists all entries in archive file.*
- [ArchEntry](#) [fileinfo](#) (const std::filesystem::path &address, const std::string &filename)  
*Returns [ArchEntry](#) for particular file or directory in archive.*
- int [libarchive\\_packing](#) (const std::filesystem::path &sourcepath, const std::filesystem::path &outpath)  
*Packs file or directory into archive.*
- int [libarchive\\_packing](#) (const std::shared\_ptr< archive > &a, const std::filesystem::path &sourcepath, const bool &rename\_source, const std::string &new\_source\_name)  
*Packs file or directory into archive.*
- [ArchiveRemoveEntry](#) [libarchive\\_remove\\_init](#) (const std::filesystem::path &sourcepath, const std::filesystem::path &outpath)  
*Initializes archive objects for removing entries from archive.*
- int [libarchive\\_remove\\_entry](#) ([ArchiveRemoveEntry](#) rm\_e, const std::vector< [ArchEntry](#) > &to\_remove)  
*Removes entry from archive.*

- void [libarchive\\_error](#) (const std::shared\_ptr< archive > &a, const std::string &message, const int &error\_↵  
number)  
*Prints libarchive error messages.*
- std::string [libarchive\\_read\\_entry\\_str](#) (archive \*a, archive\_entry \*entry)  
*Reads archived file to string.*
- int [libarchive\\_write\\_data](#) (archive \*a, const std::string &data)  
*Writes data to archive.*
- std::shared\_ptr< [ArchiveFileEntry](#) > [createArchFile](#) (const std::filesystem::path &archaddress, const la\_↵  
int64\_t &position=la\_int64\_t(0))  
*Creates [ArchiveFileEntry](#) object.*
- std::shared\_ptr< archive > [libarchive\\_read\\_init](#) (std::shared\_ptr< [ArchiveFileEntry](#) > fl)  
*Initializes archive reading.*
- std::shared\_ptr< archive > [libarchive\\_read\\_init\\_fallback](#) (std::shared\_ptr< [ArchiveFileEntry](#) > fl)  
*Initializes archive reading.*
- std::filesystem::path [libarchive\\_read\\_entry](#) (archive \*a, archive\_entry \*entry, const std::filesystem::path &out-  
folder)  
*Unpacks libarchive entry content.*
- std::shared\_ptr< archive > [libarchive\\_write\\_init](#) (const std::filesystem::path &outpath)  
*Initializes writing to archive.*
- int [writeToArchive](#) (std::shared\_ptr< archive > a, const std::filesystem::path &source, const std::filesystem::↵  
path &path\_in\_arch)  
*Writes file or directory to archive.*
- int [libarchive\\_write\\_directory](#) (archive \*a, archive\_entry \*entry, const std::filesystem::path &path\_in\_arch,  
const std::filesystem::path &source)  
*Writes directory to archive.*
- int [libarchive\\_write\\_file](#) (archive \*a, archive\_entry \*entry, const std::filesystem::path &path\_in\_arch, const  
std::filesystem::path &source)  
*Writes file to archive.*
- int [libarchive\\_write\\_data\\_from\\_file](#) (archive \*a, const std::filesystem::path &source)  
*Writes raw data from file to archive.*

### 4.18.1 Detailed Description

The [EPUBParser](#) class.

This class contains various methods for epub books parsing. In most cases you do not need to use this class directly. Use [CreateCollection](#), [RefreshCollection](#) and [BookInfo](#) instead.

### 4.18.2 Constructor & Destructor Documentation

#### 4.18.2.1 EPUBParser()

```
EPUBParser::EPUBParser (
    const std::shared_ptr< AuxFunc > & af)
```

[EPUBParser](#) constructor.

Parameters

<i>af</i>	smart pointer to <a href="#">AuxFunc</a> object.
-----------	--

### 4.18.3 Member Function Documentation

#### 4.18.3.1 epub\_book\_info()

```
std::shared_ptr< BookInfoEntry > EPUBParser::epub_book_info (
    const std::filesystem::path & filepath)
```

Returns epub book info and cover.

##### Parameters

<i>filepath</i>	absolute path to epub file.
-----------------	-----------------------------

##### Returns

Smart pointer to [BookInfoEntry](#) object.

#### 4.18.3.2 epub\_parser()

```
BookParseEntry EPUBParser::epub_parser (
    const std::filesystem::path & filepath)
```

Parses epub book.

##### Parameters

<i>filepath</i>	absolute path to epub file.
-----------------	-----------------------------

##### Returns

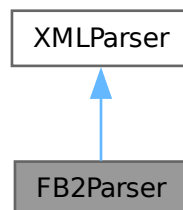
[BookParseEntry](#) object.

## 4.19 FB2Parser Class Reference

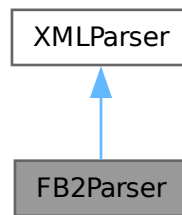
The [FB2Parser](#) class.

```
#include <FB2Parser.h>
```

Inheritance diagram for FB2Parser:



Collaboration diagram for FB2Parser:



### Public Member Functions

- [FB2Parser](#) (const std::shared\_ptr< [AuxFunc](#) > &af)  
*FB2Parser constructor.*
- [BookParseEntry fb2\\_parser](#) (const std::string &book)  
*Parses fb2 book.*
- std::shared\_ptr< [BookInfoEntry](#) > [fb2\\_book\\_info](#) (const std::string &book)  
*Returns fb2 book info and cover.*

### Public Member Functions inherited from [XMLParser](#)

- [XMLParser](#) (const std::shared\_ptr< [AuxFunc](#) > &af)  
*XMLParser constructor.*
- std::vector< [XMLTag](#) > [get\\_tag](#) (const std::string &book, const std::string &tag\_id)  
*Returns all tags with particular name.*
- std::string [get\\_book\\_encoding](#) (const std::string &book)  
*Returns [XML](#) document encoding.*
- std::string [get\\_element\\_attribute](#) (const std::string &element, const std::string &attr\_name)  
*Returns [XML](#) tag attribute if it was found.*
- std::vector< [XMLTag](#) > [listAllTags](#) (const std::string &book)  
*Parses [XML](#) document.*
- void [searchTag](#) (const std::vector< [XMLTag](#) > &list, const std::string &tag\_id, std::vector< [XMLTag](#) > &result)  
*Searches tag in tag list.*
- void [htmlSymbolsReplacement](#) (std::string &book)  
*Replaces symbols encoded by "&..." sequences.*
- void [removeAllTags](#) (std::string &book)  
*Removes all tag elements from [XML](#) document.*

#### 4.19.1 Detailed Description

The [FB2Parser](#) class.

This class contains various methods for fb2 books parsing. In most cases you do not need to use this class directly. Use [CreateCollection](#), [RefreshCollection](#) and [BookInfo](#) instead.



## 4.19.2 Constructor & Destructor Documentation

### 4.19.2.1 FB2Parser()

```
FB2Parser::FB2Parser (
    const std::shared_ptr< AuxFunc > & af)
```

[FB2Parser](#) constructor.

#### Parameters

<i>af</i>	smart pointer to <a href="#">AuxFunc</a> object.
-----------	--

## 4.19.3 Member Function Documentation

### 4.19.3.1 fb2\_book\_info()

```
std::shared_ptr< BookInfoEntry > FB2Parser::fb2_book_info (
    const std::string & book)
```

Returns fb2 book info and cover.

#### Parameters

<i>book</i>	fb2 books file content.
-------------	-------------------------

#### Returns

Smart pointer to [BookInfoEntry](#) object.

### 4.19.3.2 fb2\_parser()

```
BookParseEntry FB2Parser::fb2_parser (
    const std::string & book)
```

Parses fb2 book.

#### Note

This method can throw [MLException](#) object in case of any errors.

#### Parameters

<i>book</i>	fb2 file content.
-------------	-------------------

#### Returns

[BookParseEntry](#) object.

## 4.20 FileParseEntry Class Reference

The [FileParseEntry](#) class.

```
#include <FileParseEntry.h>
```

### Public Member Functions

- **FileParseEntry** ()  
*FileParseEntry* constructor.
- **FileParseEntry** (const [FileParseEntry](#) &other)  
*FileParseEntry* copy constructor.
- **FileParseEntry** ([FileParseEntry](#) &&other)  
*FileParseEntry* move constructor.
- **FileParseEntry** & **operator=** (const [FileParseEntry](#) &other)  
*operator =*
- **FileParseEntry** & **operator=** ([FileParseEntry](#) &&other)  
*operator =*

### Public Attributes

- std::string [file\\_rel\\_path](#)  
*Relative path to file in collection.*
- std::string [file\\_hash](#)  
*File hash sum.*
- std::vector< [BookParseEntry](#) > [books](#)  
*Contains books info.*

### 4.20.1 Detailed Description

The [FileParseEntry](#) class.

Auxiliary class, used in [CreateCollection](#), [RefreshCollection](#) and [BaseKeeper](#). Contains collection file info.

### 4.20.2 Member Data Documentation

#### 4.20.2.1 books

```
std::vector<BookParseEntry> FileParseEntry::books
```

Contains books info.

Vector containing information about books in file.

#### 4.20.2.2 file\_hash

```
std::string FileParseEntry::file_hash
```

File hash sum.

Blake-256 algorithm currently used.

#### 4.20.2.3 file\_rel\_path

```
std::string FileParseEntry::file_rel_path
```

Relative path to file in collection.

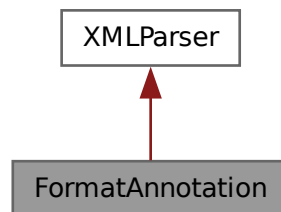
Path is relative to collection directory path.

## 4.21 FormatAnnotation Class Reference

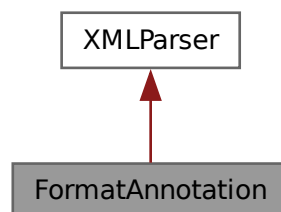
The [FormatAnnotation](#) class.

```
#include <FormatAnnotation.h>
```

Inheritance diagram for FormatAnnotation:



Collaboration diagram for FormatAnnotation:



## Public Member Functions

- [FormatAnnotation](#) (const std::shared\_ptr< [AuxFunc](#) > &af)  
*FormatAnnotation constructor.*
- void [remove\\_escape\\_sequences](#) (std::string &annotation)  
*Removes escape sequences from annotation.*
- void [replace\\_tags](#) (std::string &annotation)  
*Replaces XML tags.*
- void [final\\_cleaning](#) (std::string &annotation)  
*Cleans some sequences from annotation.*
- void [removeAllTags](#) (std::string &annotation)  
*Simply removes all XML tags.*
- void [setTagReplacementTable](#) (const std::vector< [ReplaceTagItem](#) > &replacement\_table)  
*Sets tag replacement table.*

### 4.21.1 Detailed Description

The [FormatAnnotation](#) class.

This class contains different methods for annotation processing.

### 4.21.2 Constructor & Destructor Documentation

#### 4.21.2.1 FormatAnnotation()

```
FormatAnnotation::FormatAnnotation (
    const std::shared_ptr< AuxFunc > & af)
```

[FormatAnnotation](#) constructor.

Parameters

<i>af</i>	smart pointer to <a href="#">AuxFunc</a> object.
-----------	--

### 4.21.3 Member Function Documentation

#### 4.21.3.1 final\_cleaning()

```
void FormatAnnotation::final_cleaning (
    std::string & annotation)
```

Cleans some sequences from annotation.

Replaces "three spaces" by "two spaces" at the annotation beginning, removes "\n" from annotation beginning, removes 0 - 32 [ASCII](#) symbols from the annotation end. Also replaces "\n\n\n" sequences by "\n\n" in whole annotation.

## Parameters

<i>annotation</i>	UTF-8 annotation content string.
-------------------	----------------------------------

#### 4.21.3.2 remove\_escape\_sequences()

```
void FormatAnnotation::remove_escape_sequences (
    std::string & annotation)
```

Removes escape sequences from annotation.

Removes [ASCII](#) symbols 0 to 31 (inclusive) from annotation. Only exception is symbol 9 (horizontal tab). It will be replaced by 32 (space). Also removes extra spaces from annotation beginning.

## Parameters

<i>annotation</i>	UTF-8 annotation content string.
-------------------	----------------------------------

#### 4.21.3.3 removeAllTags()

```
void FormatAnnotation::removeAllTags (
    std::string & annotation)
```

Simply removes all XML tags.

## Parameters

<i>annotation</i>	UTF-8 annotation content string.
-------------------	----------------------------------

#### 4.21.3.4 replace\_tags()

```
void FormatAnnotation::replace_tags (
    std::string & annotation)
```

Replaces XML tags.

It is highly recommended to call [setTagReplacementTable\(\)](#) method before this method call (but it is not compulsory). If tag replacement table is empty, all tags will be just removed.

## Parameters

<i>annotation</i>	UTF-8 annotation content string.
-------------------	----------------------------------

#### 4.21.3.5 setTagReplacementTable()

```
void FormatAnnotation::setTagReplacementTable (
    const std::vector< ReplaceTagItem > & replacement_table)
```

Sets tag replacement table.

If you need to replace XML tags by your own tags or text, you should call this method before [replace\\_tags\(\)](#) call to set replacement table.

## Parameters

<code>replacement_table</code>	vector containing <a href="#">ReplaceTagItem</a> objects.
--------------------------------	---

## 4.22 Genre Class Reference

The [Genre](#) class.

```
#include <Genre.h>
```

### Public Member Functions

- [Genre](#) ()  
*Genre constructor.*
- [Genre](#) (const [Genre](#) &other)  
*Genre copy constructor.*
- [Genre](#) ([Genre](#) &&other)  
*Genre move constructor.*
- [Genre](#) & **operator=** (const [Genre](#) &other)  
*operator =*
- [Genre](#) & **operator=** ([Genre](#) &&other)  
*operator =*

### Public Attributes

- std::string [genre\\_code](#)  
*fb2 genre code.*
- std::string [genre\\_name](#)  
*Translated human-readable genre name.*

### 4.22.1 Detailed Description

The [Genre](#) class.

Auxiliary class containing translated genre element (see also [AuxFunc::get\\_genre\\_list\(\)](#) and [GenreGroup](#)).

### 4.22.2 Member Data Documentation

#### 4.22.2.1 genre\_code

```
std::string Genre::genre_code
```

fb2 genre code.

This code is usually valid for epub books also.

#### 4.22.2.2 genre\_name

```
std::string Genre::genre_name
```

Translated human-readable genre name.

If translation is not available, English genre name will be set. For translations see "<share\_path>/MLBookProc/genres.csv"

## 4.23 GenreGroup Class Reference

The [GenreGroup](#) class.

```
#include <GenreGroup.h>
```

### Public Member Functions

- **GenreGroup** ()  
*GenreGroup* constructor.
- **GenreGroup** (const [GenreGroup](#) &other)  
*GenreGroup* copy constructor.
- **GenreGroup** ([GenreGroup](#) &&other)  
*GenreGroup* move constructor.
- [GenreGroup](#) & **operator=** (const [GenreGroup](#) &other)  
*operator =*
- [GenreGroup](#) & **operator=** ([GenreGroup](#) &&other)  
*operator =*

### Public Attributes

- std::string **group\_code**  
*Genre* group code name.
- std::string **group\_name**  
*Translated human-readable genre group name.*
- std::vector< [Genre](#) > **genres**  
*List of group genres (see [Genre](#)).*

### 4.23.1 Detailed Description

The [GenreGroup](#) class.

Auxiliary class containing genre group (see also [AuxFunc::get\\_genre\\_list\(\)](#)).

## 4.23.2 Member Data Documentation

### 4.23.2.1 group\_name

```
std::string GenreGroup::group_name
```

Translated human-readable genre group name.

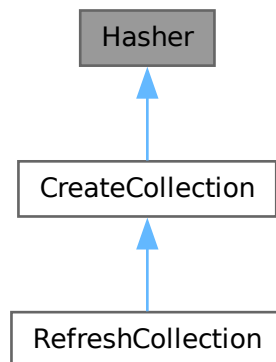
If translation is not available, English genre group name will be set. For translations see "<share\_path>/MLBookProc/genre\_groups.csv"

## 4.24 Hasher Class Reference

The [Hasher](#) class.

```
#include <Hasher.h>
```

Inheritance diagram for Hasher:



### Public Member Functions

- [Hasher](#) (const std::shared\_ptr< [AuxFunc](#) > &af)  
*Hasher constructor.*
- std::string [buf\\_hashing](#) (const std::string &buf)  
*Creates hash sum for given buffer.*
- std::string [file\\_hashing](#) (const std::filesystem::path &filepath)  
*Creates hash sum for given file.*
- void **cancelAll** ()  
*Stops all operations.*



**Protected Attributes**

- `std::atomic< bool >` [cancel](#)  
*Stops all operations if true.*
- `std::function< void()>` [stop\\_all\\_signal](#)  
*Stop signal for heir classes.*

**4.24.1 Detailed Description**

The [Hasher](#) class.

This class contains methods for hash sums creating (Blake-256 algorithm).

**4.24.2 Constructor & Destructor Documentation****4.24.2.1 Hasher()**

```
Hasher::Hasher (
    const std::shared_ptr< AuxFunc > & af)
```

[Hasher](#) constructor.

**Parameters**

<i>af</i>	smart pointer to <a href="#">AuxFunc</a> object.
-----------	--

**4.24.3 Member Function Documentation****4.24.3.1 buf\_hashing()**

```
std::string Hasher::buf_hashing (
    const std::string & buf)
```

Creates hash sum for given buffer.

**Note**

This method can throw [MLException](#) in case of error.

**Parameters**

<i>buf</i>	source buffer.
------------	----------------

**Returns**

32 bytes of hash sum value.

**4.24.3.2 file\_hashing()**

```
std::string Hasher::file_hashing (
    const std::filesystem::path & filepath)
```

Creates hash sum for given file.

**Note**

This method can throw [MLException](#) in case of error.

#### Parameters

<i>filepath</i>	absolute path to file to be hashed.
-----------------	-------------------------------------

#### Returns

32 bytes of hash sum value.

### 4.24.4 Member Data Documentation

#### 4.24.4.1 cancel

```
std::atomic<bool> Hasher::cancel [protected]
```

Stops all operations if *true*.

#### Warning

Do not call or set this variable yourself!

#### 4.24.4.2 stop\_all\_signal

```
std::function<void()> Hasher::stop_all_signal [protected]
```

Stop signal for heir classes.

#### Warning

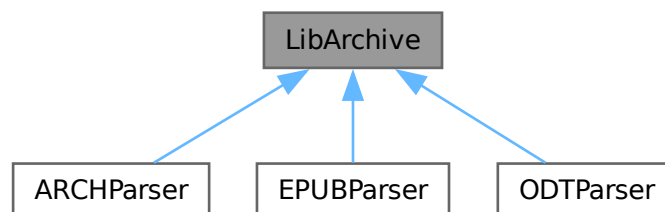
Do not call or set this variable yourself!

## 4.25 LibArchive Class Reference

The [LibArchive](#) class.

```
#include <LibArchive.h>
```

Inheritance diagram for LibArchive:



## Public Member Functions

- [LibArchive](#) (const std::shared\_ptr< [AuxFunc](#) > &af)  
*LibArchive constructor.*
- std::filesystem::path [unpackByPosition](#) (const std::filesystem::path &archaddress, const std::filesystem::path &outfolder, const [ArchEntry](#) &entry)  
*Unpacks single entry content from zip archive.*
- std::string [unpackByPositionStr](#) (const std::filesystem::path &archaddress, const [ArchEntry](#) &entry)  
*Unpacks single entry content from zip archive.*
- std::filesystem::path [unpackByFileNameStream](#) (const std::filesystem::path &archaddress, const std::filesystem::path &outfolder, const std::string &filename)  
*Unpacks entry content from archive.*
- std::string [unpackByFileNameStreamStr](#) (const std::filesystem::path &archaddress, const std::string &filename)  
*Unpacks entry content from archive.*
- int [fileNames](#) (const std::filesystem::path &filepath, std::vector< [ArchEntry](#) > &filenames)  
*Lists all entries in archive file.*
- int [fileNamesStream](#) (const std::filesystem::path &address, std::vector< [ArchEntry](#) > &filenames)  
*Lists all entries in archive file.*
- [ArchEntry](#) [fileinfo](#) (const std::filesystem::path &address, const std::string &filename)  
*Returns ArchEntry for particular file or directory in archive.*
- int [libarchive\\_packing](#) (const std::filesystem::path &sourcepath, const std::filesystem::path &outpath)  
*Packs file or directory into archive.*
- int [libarchive\\_packing](#) (const std::shared\_ptr< archive > &a, const std::filesystem::path &sourcepath, const bool &rename\_source, const std::string &new\_source\_name)  
*Packs file or directory into archive.*
- [ArchiveRemoveEntry](#) [libarchive\\_remove\\_init](#) (const std::filesystem::path &sourcepath, const std::filesystem::path &outpath)  
*Initializes archive objects for removing entries from archive.*
- int [libarchive\\_remove\\_entry](#) ([ArchiveRemoveEntry](#) rm\_e, const std::vector< [ArchEntry](#) > &to\_remove)  
*Removes entry from archive.*
- void [libarchive\\_error](#) (const std::shared\_ptr< archive > &a, const std::string &message, const int &error\_number)  
*Prints libarchive error messages.*
- std::string [libarchive\\_read\\_entry\\_str](#) (archive \*a, archive\_entry \*entry)  
*Reads archived file to string.*
- int [libarchive\\_write\\_data](#) (archive \*a, const std::string &data)  
*Writes data to archive.*
- std::shared\_ptr< [ArchiveFileEntry](#) > [createArchFile](#) (const std::filesystem::path &archaddress, const la\_int64\_t &position=la\_int64\_t(0))  
*Creates ArchiveFileEntry object.*
- std::shared\_ptr< archive > [libarchive\\_read\\_init](#) (std::shared\_ptr< [ArchiveFileEntry](#) > fl)  
*Initializes archive reading.*
- std::shared\_ptr< archive > [libarchive\\_read\\_init\\_fallback](#) (std::shared\_ptr< [ArchiveFileEntry](#) > fl)  
*Initializes archive reading.*
- std::filesystem::path [libarchive\\_read\\_entry](#) (archive \*a, archive\_entry \*entry, const std::filesystem::path &outfolder)  
*Unpacks libarchive entry content.*
- std::shared\_ptr< archive > [libarchive\\_write\\_init](#) (const std::filesystem::path &outpath)  
*Initializes writing to archive.*
- int [writeToArchive](#) (std::shared\_ptr< archive > a, const std::filesystem::path &source, const std::filesystem::path &path\_in\_arch)

*Writes file or directory to archive.*

- int [libarchive\\_write\\_directory](#) (archive \*a, archive\_entry \*entry, const std::filesystem::path &path\_in\_arch, const std::filesystem::path &source)

*Writes directory to archive.*

- int [libarchive\\_write\\_file](#) (archive \*a, archive\_entry \*entry, const std::filesystem::path &path\_in\_arch, const std::filesystem::path &source)

*Writes file to archive.*

- int [libarchive\\_write\\_data\\_from\\_file](#) (archive \*a, const std::filesystem::path &source)

*Writes raw data from file to archive.*

### 4.25.1 Detailed Description

The [LibArchive](#) class.

This class contains various methods for archives processing. Based on [libarchive](#) library.

### 4.25.2 Constructor & Destructor Documentation

#### 4.25.2.1 LibArchive()

```
LibArchive::LibArchive (
    const std::shared_ptr< AuxFunc > & af)
```

[LibArchive](#) constructor.

Parameters

<i>af</i>	smart pointer to <a href="#">AuxFunc</a> object.
-----------	--

### 4.25.3 Member Function Documentation

#### 4.25.3.1 createArchFile()

```
std::shared_ptr< ArchiveFileEntry > LibArchive::createArchFile (
    const std::filesystem::path & archaddress,
    const la_int64_t & position = la_int64_t(0))
```

Creates [ArchiveFileEntry](#) object.

[ArchiveFileEntry](#) object is used in [libarchive\\_read\\_init\(\)](#) and [libarchive\\_read\\_init\\_fallback\(\)](#) methods.

Parameters

<i>archaddress</i>	absolute path to archive to be read.
<i>position</i>	position in archive file to start reading from.

Returns

Smart pointer to [ArchiveFileEntry](#) object.

#### 4.25.3.2 fileinfo()

```
ArchEntry LibArchive::fileinfo (
    const std::filesystem::path & address,
    const std::string & filename)
```

Returns [ArchEntry](#) for particular file or directory in archive.

In case of any error [ArchEntry](#) filename will be empty.

##### Parameters

<i>address</i>	absolute path to archive.
<i>filename</i>	path in archive.

##### Returns

[ArchEntry](#) object.

#### 4.25.3.3 fileNames()

```
int LibArchive::fileNames (
    const std::filesystem::path & filepath,
    std::vector< ArchEntry > & filenames)
```

Lists all entries in archive file.

##### Warning

This method is suitable for zip archives only. For other archive types behavior is undefined.

##### Parameters

<i>filepath</i>	absolute path to zip archive.
<i>filenames</i>	vector for results.

##### Returns

1 in case of success, -1 in case of error, 0 in case archive file has not been opened.

#### 4.25.3.4 fileNamesStream()

```
int LibArchive::fileNamesStream (
    const std::filesystem::path & address,
    std::vector< ArchEntry > & filenames)
```

Lists all entries in archive file.

This method can be used with all supported archive types (see [AuxFunc::get\\_supported\\_archive\\_types\\_unpacking\(\)](#)). However for zip archives it is recommended to use [fileNames\(\)](#) method.

**Parameters**

<i>address</i>	absolute path to archive.
<i>filenames</i>	vector of results.

**Returns**

in case of succes returns ARCHIVE\_OK, libarchive error codes will be returned otherwise (see archive.h file for details).

**4.25.3.5 libarchive\_error()**

```
void LibArchive::libarchive_error (
    const std::shared_ptr< archive > & a,
    const std::string & message,
    const int & error_number)
```

Prints libarchive error messages.

**Parameters**

<i>a</i>	smart pointer to libarchive object.
<i>message</i>	extra text if needed (will be shown before libarchive error text).
<i>error_number</i>	error code.

**4.25.3.6 libarchive\_packing()** [1/2]

```
int LibArchive::libarchive_packing (
    const std::filesystem::path & sourcepath,
    const std::filesystem::path & outpath)
```

Packs file or directory into archive.

**Parameters**

<i>sourcepath</i>	absolute path to file or directory to be packed.
<i>outpath</i>	absolute path to resulting archive.

**Returns**

-100 in case if sourcepath does not exist, -200 in case of error on libarchive object creation, ARCHIVE\_OK in case of success, libarchive error code otherwise (see libarchive archive.h file for details).

**4.25.3.7 libarchive\_packing()** [2/2]

```
int LibArchive::libarchive_packing (
    const std::shared_ptr< archive > & a,
    const std::filesystem::path & sourcepath,
    const bool & rename_source,
    const std::string & new_source_name)
```

Packs file or directory into archive.

Use this method if you need source file or directory to be packed under another name.

## Parameters

<i>a</i>	smart pointer to libarchive object (see <a href="#">libarchive_write_init()</a> ).
<i>sourcepath</i>	absolute path to file or directory to be packed.
<i>rename_source</i>	if set to <i>true</i> , source name will be replaced for <b>new_source_name</b> inside the archive.
<i>new_source_name</i>	new name to be used inside the archive. Should be UTF-8 string.

## Returns

-100 in case if sourcepath does not exist, -200 in case of error on libarchive object creation, ARCHIVE\_OK in case of success, libarchive error code otherwise (see libarchive archive.h file for details).

## 4.25.3.8 libarchive\_read\_entry()

```
std::filesystem::path LibArchive::libarchive_read_entry (
    archive * a,
    archive_entry * entry,
    const std::filesystem::path & outfolder)
```

Unpacks libarchive entry content.

In most cases you do not need to use this method directly. Use [unpackByPosition\(\)](#), [unpackByPositionStr\(\)](#), [unpackByFileNameStream\(\)](#), [unpackByFileNameStreamStr\(\)](#) methods instead.

## Parameters

<i>a</i>	pointer to libarchive object.
<i>entry</i>	pointer to libarchive entry object.
<i>outfolder</i>	directory entry content to be unpacked to. If this directory does not exist, it will be created.

## Returns

Absolute path to unpacked file or directory.

## 4.25.3.9 libarchive\_read\_entry\_str()

```
std::string LibArchive::libarchive_read_entry_str (
    archive * a,
    archive_entry * entry)
```

Reads archived file to string.

If entry is not a file, empty string will be returned. In most cases you do not need to call this method directly.

## Parameters

<i>a</i>	pointer to libarchive object.
<i>entry</i>	entry to be read.

## Returns

File content.

#### 4.25.3.10 libarchive\_read\_init()

```
std::shared_ptr< archive > LibArchive::libarchive_read_init (
    std::shared_ptr< ArchiveFileEntry > fl)
```

Initializes archive reading.

This method can return *nullptr* in case of error. If this method failed, you can try to use [libarchive\\_read\\_init\\_fallback\(\)](#) instead.

##### Parameters

<i>fl</i>	smart pointer to <a href="#">ArchiveFileEntry</a> object (see <a href="#">createArchFile()</a> ).
-----------	---

##### Returns

Smart pointer to libarchive object (can be *nullptr* in case of any error).

#### 4.25.3.11 libarchive\_read\_init\_fallback()

```
std::shared_ptr< archive > LibArchive::libarchive_read_init_fallback (
    std::shared_ptr< ArchiveFileEntry > fl)
```

Initializes archive reading.

This method can return *nullptr* in case of error.

##### Parameters

<i>fl</i>	smart pointer to <a href="#">ArchiveFileEntry</a> object (see <a href="#">createArchFile()</a> ).
-----------	---

##### Returns

Smart pointer to libarchive object (can be *nullptr* in case of any error).

#### 4.25.3.12 libarchive\_remove\_entry()

```
int LibArchive::libarchive_remove_entry (
    ArchiveRemoveEntry rm_e,
    const std::vector< ArchEntry > & to_remove)
```

Removes entry from archive.

##### Parameters

<i>rm_e</i>	<a href="#">ArchiveRemoveEntry</a> got from <a href="#">libarchive_remove_init()</a> .
<i>to_remove</i>	list of entries to be removed.

##### Returns

ARCHIVE\_OK in case of success, libarchive error code otherwise (see libarchive archive.h for details).

#### 4.25.3.13 libarchive\_remove\_init()

```
ArchiveRemoveEntry LibArchive::libarchive_remove_init (
    const std::filesystem::path & sourcepath,
    const std::filesystem::path & outpath)
```

Initializes archive objects for removing entries from archive.



## Parameters

<i>sourcepath</i>	absolute path to archive entries to be removed from.
<i>outpath</i>	absolute path to write new archive without removed entries.

## Returns

[ArchiveRemoveEntry](#) object.

**4.25.3.14 libarchive\_write\_data()**

```
int LibArchive::libarchive_write_data (  
    archive * a,  
    const std::string & data)
```

Writes data to archive.

Writes raw data to archive. In most cases you do not need to call this method directly. Use [libarchive\\_write\\_directory\(\)](#) and [libarchive\\_write\\_file\(\)](#) methods instead.

## Parameters

<i>a</i>	pointer to libarchive object.
<i>data</i>	data to be written.

## Returns

libarchive error code (see libarchive archive.h for details).

**4.25.3.15 libarchive\_write\_data\_from\_file()**

```
int LibArchive::libarchive_write_data_from_file (  
    archive * a,  
    const std::filesystem::path & source)
```

Writes raw data from file to archive.

In most cases you do not need to use this method. Use [writeToArchive\(\)](#) instead.

## Parameters

<i>a</i>	pointer to libarchive object.
<i>source</i>	absolute path to source file.

## Returns

libarchive error code (see libarchive archive.h for details).

#### 4.25.3.16 libarchive\_write\_directory()

```
int LibArchive::libarchive_write_directory (
    archive * a,
    archive_entry * entry,
    const std::filesystem::path & path_in_arch,
    const std::filesystem::path & source)
```

Writes directory to archive.

In most case you do not need to use this method. Use [writeToArchive\(\)](#) instead.

##### Note

This method resolves symbolic links and processes them as underlying objects.

##### Parameters

<i>a</i>	pointer to libarchive object.
<i>entry</i>	pointer to libarchive entry object.
<i>path_in_arch</i>	path of entry in archive (must be relative, example: "my_directory/my_file").
<i>source</i>	absolute path to directory to be packed.

##### Returns

ARCHIVE\_OK in case of success, libarchive error code otherwise (see libarchive archive.h for details).

#### 4.25.3.17 libarchive\_write\_file()

```
int LibArchive::libarchive_write_file (
    archive * a,
    archive_entry * entry,
    const std::filesystem::path & path_in_arch,
    const std::filesystem::path & source)
```

Writes file to archive.

In most case you do not need to use this method. Use [writeToArchive\(\)](#) instead.

##### Parameters

<i>a</i>	pointer to libarchive object.
<i>entry</i>	pointer to libarchive entry object.
<i>path_in_arch</i>	path of entry in archive (must be relative, example: "my_directory/my_file").
<i>source</i>	absolute path to directory to be packed.

##### Returns

ARCHIVE\_OK in case of success, libarchive error code otherwise (see libarchive archive.h for details).

#### 4.25.3.18 libarchive\_write\_init()

```
std::shared_ptr< archive > LibArchive::libarchive_write_init (
    const std::filesystem::path & outpath)
```

Initializes writing to archive.

##### Warning

If resulting archive path already exists, it will be overwritten.

## Parameters

<i>outpath</i>	path to resulting archive.
----------------	----------------------------

## Returns

Smart pointer to libarchive object (can be *nullptr* in case of any error).

**4.25.3.19 unpackByFileNameStream()**

```
std::filesystem::path LibArchive::unpackByFileNameStream (  
    const std::filesystem::path & archaddress,  
    const std::filesystem::path & outfolder,  
    const std::string & filename)
```

Unpacks entry content from archive.

This method is suitable for any supported types of archives (see [AuxFunc::get\\_supported\\_archive\\_types\\_unpacking\(\)](#)). However for zip archives it is recommended to use [unpackByPosition\(\)](#) and [unpackByPositionStr\(\)](#) methods (they are a little bit faster).

## Parameters

<i>archaddress</i>	absolute path to archive.
<i>outfolder</i>	absolute path to directory, entry to be unpacked to. If directory does not exist, it will be created.
<i>filename</i>	file or directory name in archive.

## Returns

Absolute path to unpacked file or directory.

**4.25.3.20 unpackByFileNameStreamStr()**

```
std::string LibArchive::unpackByFileNameStreamStr (  
    const std::filesystem::path & archaddress,  
    const std::string & filename)
```

Unpacks entry content from archive.

If entry is a file, unpacks it and returns file content. If entry is a directory, returns empty string. This method is suitable for any supported types of archives (see [AuxFunc::get\\_supported\\_archive\\_types\\_unpacking\(\)](#)). However for zip archives it is recommended to use [unpackByPosition\(\)](#) and [unpackByPositionStr\(\)](#) methods (they are a little bit faster).

## Parameters

<i>archaddress</i>	absolute path to archive.
<i>filename</i>	file or directory name in archive.

## Returns

Unpacked file content or empty string.

#### 4.25.3.21 unpackByPosition()

```
std::filesystem::path LibArchive::unpackByPosition (
    const std::filesystem::path & archaddress,
    const std::filesystem::path & outfolder,
    const ArchEntry & entry)
```

Unpacks single entry content from zip archive.

Access to entry is carried out by its absolute position in zip file. It is recommended to use this method for fast unpacking of single file or directory from zip archive.

##### Warning

This method should be used for zip archives only!

##### Parameters

<i>archaddress</i>	absolute path to zip archive.
<i>outfolder</i>	absolute path to directory archive entry content to be unpacked to. If directory does not exist, it will be created.
<i>entry</i>	<a href="#">ArchEntry</a> object, obtained by <a href="#">fileNames()</a> , <a href="#">fileNamesStream()</a> or <a href="#">fileinfo()</a> methods.

##### Returns

Absolute path to unpacked file or directory.

#### 4.25.3.22 unpackByPositionStr()

```
std::string LibArchive::unpackByPositionStr (
    const std::filesystem::path & archaddress,
    const ArchEntry & entry)
```

Unpacks single entry content from zip archive.

If entry is a file, unpacks it and returns file content. If entry is a directory, returns empty string. Access to entry is carried out by its absolute position in zip file. It is recommended to use this method for fast unpacking of single file from zip archive.

##### Warning

This method should be used for zip archives only!

##### Parameters

<i>archaddress</i>	absolute path to zip archive.
<i>entry</i>	<a href="#">ArchEntry</a> object, obtained by <a href="#">fileNames()</a> , <a href="#">fileNamesStream()</a> or <a href="#">fileinfo()</a> methods.

##### Returns

Unpacked file content or empty string.

### 4.25.3.23 writeToArchive()

```
int LibArchive::writeToArchive (
    std::shared_ptr< archive > a,
    const std::filesystem::path & source,
    const std::filesystem::path & path_in_arch)
```

Writes file or directory to archive.

#### Note

This method resolves symbolic links and processes them as underlying objects.

#### Parameters

<i>a</i>	smart pointer to libarchive object (see <a href="#">libarchive_write_init()</a> ).
<i>source</i>	absolute path to file or directory to be packed.
<i>path_in_arch</i>	path of entry in archive (must be relative, example: my_directory/my_file).

#### Returns

ARCHIVE\_OK in case of success, libarchive error code otherwise (see libarchive archive.h for details).

## 4.26 MLException Class Reference

The [MLException](#) class.

```
#include <MLException.h>
```

### Public Member Functions

- **MLException ()**  
*MLException* constructor.
- **MLException** (const std::string &msg)  
*MLException* constructor.
- **MLException** (const [MLException](#) &other)  
*MLException* copy constructor.
- **MLException** ([MLException](#) &&other)  
*MLException* move constructor.
- **MLException & operator=** (const [MLException](#) &other)  
*operator =*
- **MLException & operator=** ([MLException](#) &&other)  
*operator =*
- **operator bool ()**  
Returns true if *MLException* contains message.
- std::string **what ()**  
Returns error message.

### 4.26.1 Detailed Description

The [MException](#) class.

This class is used as exception to indicate various errors.

### 4.26.2 Constructor & Destructor Documentation

#### 4.26.2.1 MException()

```
MException::MException (
    const std::string & msg)
```

[MException](#) constructor.

Parameters

<i>msg</i>	message to be printed.
------------	------------------------

### 4.26.3 Member Function Documentation

#### 4.26.3.1 what()

```
std::string MException::what ()
```

Returns error message.

Returns

Error info message.

## 4.27 NotesBaseEntry Class Reference

The [NotesBaseEntry](#) class.

```
#include <NotesBaseEntry.h>
```

### Public Member Functions

- **NotesBaseEntry ()**  
*NotesBaseEntry* constructor.
- **NotesBaseEntry** (const std::string &[collection\\_name](#), const std::filesystem::path &[book\\_file\\_full\\_path](#), const std::string &[book\\_path](#))  
*NotesBaseEntry* constructor.
- **NotesBaseEntry** (const [NotesBaseEntry](#) &other)  
*NotesBaseEntry* copy constructor.
- **NotesBaseEntry** ([NotesBaseEntry](#) &&other)  
*NotesBaseEntry* move constructor.
- **NotesBaseEntry** & **operator=** (const [NotesBaseEntry](#) &other)  
*operator =*
- **NotesBaseEntry** & **operator=** ([NotesBaseEntry](#) &&other)  
*operator =*
- bool **operator==** (const [NotesBaseEntry](#) &other) const  
*operator ==*

## Public Attributes

- `std::string` **collection\_name**  
*Collection name.*
- `std::filesystem::path` **book\_file\_full\_path**  
*Absolute path to book file.*
- `std::string` **book\_path**  
*Relative path to book in file (in case of archive, empty otherwise).*
- `std::filesystem::path` **note\_file\_full\_path**  
*Absolute path to note file.*

### 4.27.1 Detailed Description

The [NotesBaseEntry](#) class.

Auxiliary class containing note info. In most cases you do not need to create [NotesBaseEntry](#) object yourself (see [NotesKeeper](#)).

### 4.27.2 Constructor & Destructor Documentation

#### 4.27.2.1 NotesBaseEntry()

```
NotesBaseEntry::NotesBaseEntry (
    const std::string & collection_name,
    const std::filesystem::path & book_file_full_path,
    const std::string & book_path)
```

[NotesBaseEntry](#) constructor.

#### Parameters

<i>collection_name</i>	collection name.
<i>book_file_full_path</i>	absolute path to book file.
<i>book_path</i>	relative path to book in file (in case of archive, empty otherwise).

## 4.28 NotesKeeper Class Reference

The [NotesKeeper](#) class.

```
#include <NotesKeeper.h>
```



## Public Member Functions

- [NotesKeeper](#) (const std::shared\_ptr< [AuxFunc](#) > &af)  
*NotesKeeper constructor.*
- virtual ~[NotesKeeper](#) ()  
*NotesKeeper destructor.*
- void [loadBase](#) ()  
*Loads notes base to memory.*
- void [editNote](#) (const [NotesBaseEntry](#) &nbe, const std::string &note)  
*Edits note.*
- [NotesBaseEntry](#) [getNote](#) (const std::string &collection\_name, const std::filesystem::path &book\_file\_full\_path, const std::string &book\_path)  
*Gets NotesBaseEntry object from base or creates it.*
- std::string [readNote](#) (const [NotesBaseEntry](#) &nbe)  
*Returns content of note file.*
- std::string [readNoteText](#) (const [NotesBaseEntry](#) &nbe)  
*Returns note text (if any).*
- void [removeNotes](#) (const [NotesBaseEntry](#) &nbe, const std::filesystem::path &reserve\_directory, const bool &make\_reserve)  
*Removes notes.*
- void [removeCollection](#) (const std::string &collection\_name, const std::filesystem::path &reserve\_directory, const bool &make\_reserve)  
*Removes notes for all books in particular collection.*
- void [refreshCollection](#) (const std::string &collection\_name, const std::filesystem::path &reserve\_directory, const bool &make\_reserve)  
*Compares notes base and collection base and removes notes for absent books.*
- std::vector< [NotesBaseEntry](#) > [getNotesForCollection](#) (const std::string &collection\_name)  
*Returns all notes for particular collection.*

### 4.28.1 Detailed Description

The [NotesKeeper](#) class.

This class contains various methods for collections notes operating. It is recommended to start from [loadBase\(\)](#) method. To create new note call [getNote\(\)](#) and [editNote\(\)](#) methods. [getNote\(\)](#) method also can be used to obtain note for particular book in collection. [editNote\(\)](#) method can be used to remove note.

### 4.28.2 Constructor & Destructor Documentation

#### 4.28.2.1 NotesKeeper()

```
NotesKeeper::NotesKeeper (
    const std::shared_ptr< AuxFunc > & af)
```

[NotesKeeper](#) constructor.

#### Parameters

<i>af</i>	smart pointer to <a href="#">AuxFunc</a> object.
-----------	--

### 4.28.3 Member Function Documentation

#### 4.28.3.1 editNote()

```
void NotesKeeper::editNote (
    const NotesBaseEntry & nbe,
    const std::string & note)
```

Edits note.

If note file does not exist, this method will creat it and add [NotesBaseEntry](#) object to base. If note file exists, it will be overwritten. If note string is empty, note file will be removed, and [NotesBaseEntry](#) object will be removed from base.

##### Parameters

<i>nbe</i>	<a href="#">NotesBaseEntry</a> object (see <a href="#">getNote()</a> and <a href="#">getNotesForCollection()</a> ).
<i>note</i>	note text.

#### 4.28.3.2 getNote()

```
NotesBaseEntry NotesKeeper::getNote (
    const std::string & collection_name,
    const std::filesystem::path & book_file_full_path,
    const std::string & book_path)
```

Gets [NotesBaseEntry](#) object from base or creates it.

If note exists, returns its [NotesBaseEntry](#) object. If note does not exist, creates new [NotesBaseEntry](#) object.

##### Parameters

<i>collection_name</i>	collection name, book came from.
<i>book_file_full_path</i>	book file absolute path.
<i>book_path</i>	book path in file (if any, empty string otherwise).

##### Returns

[NotesBaseEntry](#) object for note.

#### 4.28.3.3 getNotesForCollection()

```
std::vector< NotesBaseEntry > NotesKeeper::getNotesForCollection (
    const std::string & collection_name)
```

Returns all notes for particular collection.

##### Parameters

<i>collection_name</i>	collection name.
------------------------	------------------

##### Returns

Vector of [NotesBaseEntry](#) objects.

#### 4.28.3.4 loadBase()

```
void NotesKeeper::loadBase ()
```

Loads notes base to memory.

This method should be called before any other methods of this class.

#### 4.28.3.5 readNote()

```
std::string NotesKeeper::readNote (
    const NotesBaseEntry & nbe)
```

Returns content of note file.

Note file contains header and note text. Header contains collection name, book file absolute path, book path in file (if any). Header is separated from note text by "\n\n" sequence.

##### Parameters

<i>nbe</i>	NotesBaseEntry object (see <a href="#">getNote()</a> and <a href="#">getNotesForCollection()</a> ).
------------	---

##### Returns

String containing note file raw content.

#### 4.28.3.6 readNoteText()

```
std::string NotesKeeper::readNoteText (
    const NotesBaseEntry & nbe)
```

Returns note text (if any).

##### Parameters

<i>nbe</i>	NotesBaseEntry object (see <a href="#">getNote()</a> and <a href="#">getNotesForCollection()</a> ).
------------	---

##### Returns

String containing note text.

#### 4.28.3.7 refreshCollection()

```
void NotesKeeper::refreshCollection (
    const std::string & collection_name,
    const std::filesystem::path & reserve_directory,
    const bool & make_reserve)
```

Compares notes base and collection base and removes notes for absent books.

## Parameters

<i>collection_name</i>	collection to compare bases.
<i>reserve_directory</i>	absolute path to directory for reserve copies of notes to be removed. If directory does not exist, it will be created.
<i>make_reserve</i>	if set to <i>true</i> , <a href="#">refreshCollection()</a> will create reserve copies of notes to be removed.

**4.28.3.8 removeCollection()**

```
void NotesKeeper::removeCollection (
    const std::string & collection_name,
    const std::filesystem::path & reserve_directory,
    const bool & make_reserve)
```

Removes notes for all books in particular collection.

## Parameters

<i>collection_name</i>	collection name.
<i>reserve_directory</i>	absolute path to directory for reserve copies of notes to be removed. If directory does not exist, it will be created.
<i>make_reserve</i>	if set to <i>true</i> , <a href="#">removeCollection()</a> will create reserve copies of notes to be removed.

**4.28.3.9 removeNotes()**

```
void NotesKeeper::removeNotes (
    const NotesBaseEntry & nbe,
    const std::filesystem::path & reserve_directory,
    const bool & make_reserve)
```

Removes notes.

It is recommended to use this method after book removing from collection.

## Warning

If *nbe* parametr **book\_file\_full\_path** contains path to rar archive, this method will remove notes for all books in archive.

## Parameters

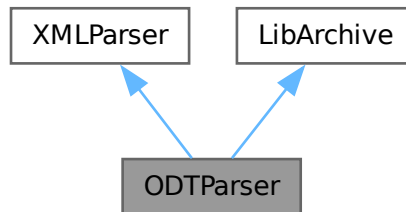
<i>nbe</i>	<a href="#">NotesBaseEntry</a> object (see <a href="#">getNote()</a> and <a href="#">getNotesForCollection()</a> ).
<i>reserve_directory</i>	absolute path to directory for reserve copies of notes to be removed. If directory does not exist, it will be created.
<i>make_reserve</i>	if set to <i>true</i> , <a href="#">removeNotes()</a> will create reserve copies of notes to be removed.

## 4.29 ODTParser Class Reference

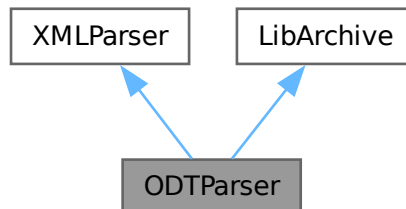
The [ODTParser](#) class.

```
#include <ODTParser.h>
```

Inheritance diagram for ODTParser:



Collaboration diagram for ODTParser:



### Public Member Functions

- [ODTParser](#) (const std::shared\_ptr< [AuxFunc](#) > &af)  
*ODTParser constructor.*
- virtual ~**ODTParser** ()  
*ODTParser destructor.*
- [BookParseEntry odtParser](#) (const std::filesystem::path &odt\_path)  
*Parses odt files.*
- std::shared\_ptr< [BookInfoEntry](#) > [odtBookInfo](#) (const std::filesystem::path &odt\_path)  
*Gets some extra info from odt files.*

## Public Member Functions inherited from XMLParser

- [XMLParser](#) (const std::shared\_ptr< [AuxFunc](#) > &af)  
*XMLParser constructor.*
- std::vector< [XMLTag](#) > [get\\_tag](#) (const std::string &book, const std::string &tag\_id)  
*Returns all tags with particular name.*
- std::string [get\\_book\\_encoding](#) (const std::string &book)  
*Returns [XML](#) document encoding.*
- std::string [get\\_element\\_attribute](#) (const std::string &element, const std::string &attr\_name)  
*Returns [XML](#) tag attribute if it was found.*
- std::vector< [XMLTag](#) > [listAllTags](#) (const std::string &book)  
*Parses [XML](#) document.*
- void [searchTag](#) (const std::vector< [XMLTag](#) > &list, const std::string &tag\_id, std::vector< [XMLTag](#) > &result)  
*Searches tag in tag list.*
- void [htmlSymbolsReplacement](#) (std::string &book)  
*Replaces symbols encoded by "&..." sequences.*
- void [removeAllTags](#) (std::string &book)  
*Removes all tag elements from [XML](#) document.*

## Public Member Functions inherited from LibArchive

- [LibArchive](#) (const std::shared\_ptr< [AuxFunc](#) > &af)  
*LibArchive constructor.*
- std::filesystem::path [unpackByPosition](#) (const std::filesystem::path &archaddress, const std::filesystem::path &outfolder, const [ArchEntry](#) &entry)  
*Unpacks single entry content from zip archive.*
- std::string [unpackByPositionStr](#) (const std::filesystem::path &archaddress, const [ArchEntry](#) &entry)  
*Unpacks single entry content from zip archive.*
- std::filesystem::path [unpackByFileNameStream](#) (const std::filesystem::path &archaddress, const std::filesystem::path &outfolder, const std::string &filename)  
*Unpacks entry content from archive.*
- std::string [unpackByFileNameStreamStr](#) (const std::filesystem::path &archaddress, const std::string &filename)  
*Unpacks entry content from archive.*
- int [fileNames](#) (const std::filesystem::path &filepath, std::vector< [ArchEntry](#) > &filenames)  
*Lists all entries in archive file.*
- int [fileNamesStream](#) (const std::filesystem::path &address, std::vector< [ArchEntry](#) > &filenames)  
*Lists all entries in archive file.*
- [ArchEntry](#) [fileinfo](#) (const std::filesystem::path &address, const std::string &filename)  
*Returns [ArchEntry](#) for particular file or directory in archive.*
- int [libarchive\\_packing](#) (const std::filesystem::path &sourcepath, const std::filesystem::path &outpath)  
*Packs file or directory into archive.*
- int [libarchive\\_packing](#) (const std::shared\_ptr< archive > &a, const std::filesystem::path &sourcepath, const bool &rename\_source, const std::string &new\_source\_name)  
*Packs file or directory into archive.*
- [ArchiveRemoveEntry](#) [libarchive\\_remove\\_init](#) (const std::filesystem::path &sourcepath, const std::filesystem::path &outpath)  
*Initializes archive objects for removing entries from archive.*
- int [libarchive\\_remove\\_entry](#) ([ArchiveRemoveEntry](#) rm\_e, const std::vector< [ArchEntry](#) > &to\_remove)  
*Removes entry from archive.*

- void [libarchive\\_error](#) (const std::shared\_ptr< archive > &a, const std::string &message, const int &error\_↵  
number)  
*Prints libarchive error messages.*
- std::string [libarchive\\_read\\_entry\\_str](#) (archive \*a, archive\_entry \*entry)  
*Reads archived file to string.*
- int [libarchive\\_write\\_data](#) (archive \*a, const std::string &data)  
*Writes data to archive.*
- std::shared\_ptr< [ArchiveFileEntry](#) > [createArchFile](#) (const std::filesystem::path &archaddress, const la\_↵  
int64\_t &position=la\_int64\_t(0))  
*Creates [ArchiveFileEntry](#) object.*
- std::shared\_ptr< archive > [libarchive\\_read\\_init](#) (std::shared\_ptr< [ArchiveFileEntry](#) > fl)  
*Initializes archive reading.*
- std::shared\_ptr< archive > [libarchive\\_read\\_init\\_fallback](#) (std::shared\_ptr< [ArchiveFileEntry](#) > fl)  
*Initializes archive reading.*
- std::filesystem::path [libarchive\\_read\\_entry](#) (archive \*a, archive\_entry \*entry, const std::filesystem::path &out-  
folder)  
*Unpacks libarchive entry content.*
- std::shared\_ptr< archive > [libarchive\\_write\\_init](#) (const std::filesystem::path &outpath)  
*Initializes writing to archive.*
- int [writeToArchive](#) (std::shared\_ptr< archive > a, const std::filesystem::path &source, const std::filesystem::↵  
path &path\_in\_arch)  
*Writes file or directory to archive.*
- int [libarchive\\_write\\_directory](#) (archive \*a, archive\_entry \*entry, const std::filesystem::path &path\_in\_arch,  
const std::filesystem::path &source)  
*Writes directory to archive.*
- int [libarchive\\_write\\_file](#) (archive \*a, archive\_entry \*entry, const std::filesystem::path &path\_in\_arch, const  
std::filesystem::path &source)  
*Writes file to archive.*
- int [libarchive\\_write\\_data\\_from\\_file](#) (archive \*a, const std::filesystem::path &source)  
*Writes raw data from file to archive.*

## 4.29.1 Detailed Description

The [ODTParser](#) class.

This class contains methods for odt files processing. In most cases you do not need to use this class directly. Use [CreateCollection](#), [RefreshCollection](#) and [BookInfo](#) instead.

## 4.29.2 Constructor & Destructor Documentation

### 4.29.2.1 ODTParser()

```
ODTParser::ODTParser (
    const std::shared_ptr< AuxFunc > & af)
```

[ODTParser](#) constructor.

Parameters

<i>af</i>	smart pointer to <a href="#">AuxFunc</a> object.
-----------	--

### 4.29.3 Member Function Documentation

#### 4.29.3.1 odtBookInfo()

```
std::shared_ptr< BookInfoEntry > ODTParser::odtBookInfo (
    const std::filesystem::path & odt_path)
```

Gets some extra info from odt files.

This method can be used to obtain odt file cover and some other info (see [BookInfoEntry](#)) if such info is available.

##### Note

This method can throw [MLException](#) in case of some errors.

##### Parameters

<i>odt_path</i>	absolute path to odt file.
-----------------	----------------------------

##### Returns

Smart pointer to [BookInfoEntry](#) object.

#### 4.29.3.2 odtParser()

```
BookParseEntry ODTParser::odtParser (
    const std::filesystem::path & odt_path)
```

Parses odt files.

This method can be used to obtain information from odt files.

##### Note

This method can throw [MLException](#) in case of some errors.

##### Parameters

<i>odt_path</i>	absolute path to odt file.
-----------------	----------------------------

##### Returns

[BookParseEntry](#) object.

## 4.30 OmpLockGuard Class Reference

The [OmpLockGuard](#) class.

```
#include <OmpLockGuard.h>
```



**Public Member Functions**

- [OmpLockGuard](#) (omp\_lock\_t &omp\_mtx)  
*OmpLockGuard* constructor.
- virtual `~OmpLockGuard ()`  
*OmpLockGuard* destructor.

**4.30.1 Detailed Description**

The [OmpLockGuard](#) class.

Auxiliary class. Locks omp\_lock\_t variable on creation and unlocks it on destruction.

**4.30.2 Constructor & Destructor Documentation****4.30.2.1 OmpLockGuard()**

```
OmpLockGuard::OmpLockGuard (
    omp_lock_t & omp_mtx)
```

[OmpLockGuard](#) constructor.

**Warning**

**omp\_mtx** must be initialized (see [omp\\_init\\_lock](#)).

**Parameters**

<i>omp_mtx</i>	omp_lock_t variable.
----------------	----------------------

**4.31 OpenBook Class Reference**

The [OpenBook](#) class.

```
#include <OpenBook.h>
```

**Public Member Functions**

- [OpenBook](#) (const std::shared\_ptr< [AuxFunc](#) > &af)  
*OpenBook* constructor.
- std::filesystem::path [open\\_book](#) (const [BookBaseEntry](#) &bbe, const bool &copy, const std::filesystem::path &copy\_path, const bool &find\_fbd, std::function< void(const std::filesystem::path &path)> open\_callback)  
*Opens book.*

### 4.31.1 Detailed Description

The [OpenBook](#) class.

This class contains methods for books "opening".

### 4.31.2 Constructor & Destructor Documentation

#### 4.31.2.1 OpenBook()

```
OpenBook::OpenBook (
    const std::shared_ptr< AuxFunc > & af)
```

[OpenBook](#) constructor.

Parameters

<i>af</i>	smart pointer to <a href="#">AuxFunc</a> object.
-----------	--

### 4.31.3 Member Function Documentation

#### 4.31.3.1 open\_book()

```
std::filesystem::path OpenBook::open_book (
    const BookBaseEntry & bbe,
    const bool & copy,
    const std::filesystem::path & copy_path,
    const bool & find_fbd,
    std::function< void(const std::filesystem::path &path)> open_callback)
```

Opens book.

If book is in archive, unpacks book and returns absolute path to unpacked file. Otherwise returns absolute path to book file.

If **copy** is set to *true* and **copy\_path** is not empty, creates directory on **copy\_path** and copies book to it.

If **find\_fbd** is set to *true*, will try to find and open fbd file instead of book.

If **open\_callback** is not *nullptr*, calls it.

Note

This method can throw [MLException](#) in case of errors.

Parameters

<i>bbe</i>	<a href="#">BookBaseEntry</a> object.
<i>copy</i>	if set to <i>true</i> , copy of book file will be created.
<i>copy_path</i>	absolute path to directory book to be copied to.
<i>find_fbd</i>	if set to <i>true</i> , this method will try to find and open fbd file instead of book.
<i>open_callback</i>	method to be called at the end of all operations. <b>path</b> argument is an absolute path to method work result.

Returns

Absolute path to book to be opened.

## 4.32 PaperBookInfoEntry Class Reference

The [PaperBookInfoEntry](#) class.

```
#include <PaperBookInfoEntry.h>
```

### Public Member Functions

- **PaperBookInfoEntry** ()  
*PaperBookInfoEntry* constructor.
- **PaperBookInfoEntry** (const [PaperBookInfoEntry](#) &other)  
*PaperBookInfoEntry* copy constructor.
- **PaperBookInfoEntry** ([PaperBookInfoEntry](#) &&other)  
*PaperBookInfoEntry* move constructor.
- **PaperBookInfoEntry** & **operator=** (const [PaperBookInfoEntry](#) &other)  
*operator =*
- **PaperBookInfoEntry** & **operator=** ([PaperBookInfoEntry](#) &&other)  
*operator =*

### Public Attributes

- bool **available** = false  
*If paper book info is available, will be set to true.*
- std::string **book\_name**  
*Paper book name.*
- std::string **publisher**  
*Paper book publisher.*
- std::string **city**  
*City where paper book was published.*
- std::string **year**  
*Year of paper book publishing.*
- std::string **isbn**  
*ISBN of paper book.*

### 4.32.1 Detailed Description

The [PaperBookInfoEntry](#) class.

Auxiliary class containing some information about paper book source (if any).

## 4.33 PDFParser Class Reference

The [PDFParser](#) class.

```
#include <PDFParser.h>
```

## Public Member Functions

- [PDFParser](#) (const std::shared\_ptr< [AuxFunc](#) > &af)  
*PDFParser constructor.*
- [BookParseEntry pdf\\_parser](#) (const std::string &file)  
*Parses pdf file.*
- std::shared\_ptr< [BookInfoEntry](#) > [pdf\\_annotation\\_n\\_cover](#) (const std::string &file, const double &x\_dpi, const double &y\_dpi)  
*Returns pdf book annotation and cover.*

### 4.33.1 Detailed Description

The [PDFParser](#) class.

This class contains methods for pdf book parsing, annotations and covers obtaining. In most cases you do not need to use this class directly. Use [CreateCollection](#), [RefreshCollection](#) and [BookInfo](#) instead.

### 4.33.2 Constructor & Destructor Documentation

#### 4.33.2.1 PDFParser()

```
PDFParser::PDFParser (
    const std::shared_ptr< AuxFunc > & af)
```

[PDFParser](#) constructor.

Parameters

<i>af</i>	smart pointer to <a href="#">AuxFunc</a> object.
-----------	--

### 4.33.3 Member Function Documentation

#### 4.33.3.1 pdf\_annotation\_n\_cover()

```
std::shared_ptr< BookInfoEntry > PDFParser::pdf_annotation_n_cover (
    const std::string & file,
    const double & x_dpi,
    const double & y_dpi)
```

Returns pdf book annotation and cover.

Parameters

<i>file</i>	pdf file content.
<i>x_dpi</i>	horizontal <a href="#">DPI</a> .
<i>y_dpi</i>	vertical <a href="#">DPI</a> .

Returns

Smart pointer to [BookInfoEntry](#) object.

#### 4.33.3.2 pdf\_parser()

```
BookParseEntry PDFParser::pdf_parser (
    const std::string & file)
```

Parses pdf file.

## Parameters

<i>file</i>	pdf file content.
-------------	-------------------

## Returns

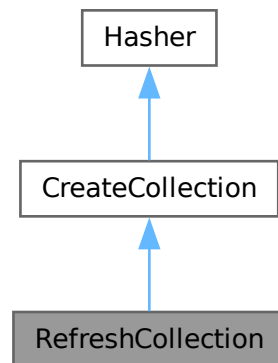
[BookParseEntry](#) object.

## 4.34 RefreshCollection Class Reference

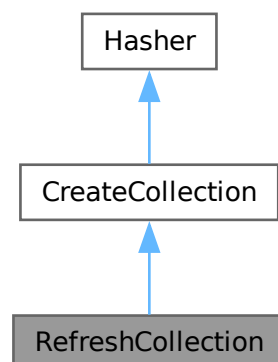
The [RefreshCollection](#) class.

```
#include <RefreshCollection.h>
```

Inheritance diagram for RefreshCollection:



Collaboration diagram for RefreshCollection:



## Public Member Functions

- [RefreshCollection](#) (const std::shared\_ptr< [AuxFunc](#) > &af, const std::string &collection\_name, const int &num\_threads, const bool &remove\_empty, const bool &refresh\_bookmarks, const bool &fast\_refresh, const std::shared\_ptr< [BookMarks](#) > &bookmarks)  
*RefreshCollection constructor.*
- virtual ~[RefreshCollection](#) ()  
*RefreshCollection destructor.*
- void [refreshCollection](#) ()  
*Refreshes whole collection.*
- void [refreshFile](#) (const [BookBaseEntry](#) &bbe)  
*Refreshes iformation about particular file.*
- bool [editBook](#) (const [BookBaseEntry](#) &bbe\_old, const [BookBaseEntry](#) &bbe\_new)  
*Replaces information in database.*
- bool [refreshBook](#) (const [BookBaseEntry](#) &bbe)  
*Refreshes information in database about particular book.*
- void [set\\_rar\\_support](#) (const bool &rar\_support)  
*Enables support of rar archives.*

## Public Member Functions inherited from [CreateCollection](#)

- [CreateCollection](#) (const std::shared\_ptr< [AuxFunc](#) > &af, const std::filesystem::path &collection\_path, const std::filesystem::path &books\_path, const bool &rar\_support, const int &num\_threads)  
*CreateCollection constructor.*
- virtual ~[CreateCollection](#) ()  
*CreateCollection destructor.*
- void [createCollection](#) ()  
*Starts collection creation.*

## Public Member Functions inherited from [Hasher](#)

- [Hasher](#) (const std::shared\_ptr< [AuxFunc](#) > &af)  
*Hasher constructor.*
- std::string [buf\\_hashing](#) (const std::string &buf)  
*Creates hash sum for given buffer.*
- std::string [file\\_hashing](#) (const std::filesystem::path &filepath)  
*Creates hash sum for given file.*
- void [cancelAll](#) ()  
*Stops all operations.*

## Public Attributes

- std::function< void(const double &total\_hash)> [total\\_bytes\\_to\\_hash](#)  
*"Total bytes to hash" signal.*
- std::function< void(const double &hashed)> [bytes\\_hashed](#)  
*"Total bytes hashed" signal.*

## Public Attributes inherited from [CreateCollection](#)

- `std::function< void()>` [pulse](#)  
*"Pulse" callback.*
- `std::function< void(const double &)>` [signal\\_total\\_bytes](#)  
*"Total bytes" callback.*
- `std::function< void(const double &progress)>` [progress](#)  
*"Progress" callback.*

## Additional Inherited Members

## Protected Member Functions inherited from [CreateCollection](#)

- [CreateCollection](#) (const `std::shared_ptr< AuxFunc >` &af, const int &num\_threads)  
*[CreateCollection](#) constructor.*
- void [threadRegulator](#) ()  
*Threads regulator.*
- void [openBaseFile](#) ()  
*Opens database file for writing.*
- void [closeBaseFile](#) ()  
*Finishes database writing.*
- void [write\\_file\\_to\\_base](#) (const [FileParseEntry](#) &fe)  
*Writes file data to database.*

## Protected Attributes inherited from [CreateCollection](#)

- `std::filesystem::path` [base\\_path](#)  
*Absolute path to database.*
- `std::filesystem::path` [books\\_path](#)  
*Absolute path to books directory.*
- bool [rar\\_support](#) = false  
*If true, rar archives will be processed, otherwise - not.*
- `std::vector< std::tuple< std::filesystem::path, std::string > >` [already\\_hashed](#)  
*Hashed files.*
- `std::vector< std::filesystem::path >` [need\\_to\\_parse](#)  
*"Need to parse" vector.*
- `std::atomic< double >` [current\\_bytes](#)  
*Keeps quantity of bytes have been processed.*

## Protected Attributes inherited from [Hasher](#)

- `std::atomic< bool >` [cancel](#)  
*Stops all operations if true.*
- `std::function< void()>` [stop\\_all\\_signal](#)  
*Stop signal for heir classes.*

### 4.34.1 Detailed Description

The [RefreshCollection](#) class.

This class contains various methods for collection database refreshing in case of any changes were made to collection files.

### 4.34.2 Constructor & Destructor Documentation

#### 4.34.2.1 RefreshCollection()

```
RefreshCollection::RefreshCollection (
    const std::shared_ptr< AuxFunc > & af,
    const std::string & collection_name,
    const int & num_threads,
    const bool & remove_empty,
    const bool & refresh_bookmarks,
    const bool & fast_refresh,
    const std::shared_ptr< BookMarks > & bookmarks)
```

[RefreshCollection](#) constructor.

#### Note

See also [set\\_rar\\_support\(\)](#) method.

#### Parameters

<i>af</i>	smart pointer to <a href="#">AuxFunc</a> object.
<i>collection_name</i>	collection name.
<i>num_threads</i>	number of threads to be used (see also <a href="#">CreateCollection::CreateCollection()</a> ).
<i>remove_empty</i>	if <i>true</i> , empty directories and files will be removed.
<i>fast_refresh</i>	if <i>true</i> , file hashes calculations will not be carried out (hashes of all collection files will be recalculated otherwise).
<i>refresh_bookmarks</i>	if <i>true</i> , bookmarks pointing to absent books will be removed.
<i>bookmarks</i>	smart pointer to <a href="#">BookMarks</a> object.

### 4.34.3 Member Function Documentation

#### 4.34.3.1 editBook()

```
bool RefreshCollection::editBook (
    const BookBaseEntry & bbe_old,
    const BookBaseEntry & bbe_new)
```

Replaces information in database.

Use this method, if you need to edit database entries manually.



## Parameters

<i>bbe_old</i>	existing <a href="#">BookBaseEntry</a> (see <a href="#">BaseKeeper::searchBook()</a> ).
<i>bbe_new</i>	<a href="#">BookBaseEntry</a> containing new information.

## Returns

Returns *true*, if operation has been successful.

**4.34.3.2 refreshBook()**

```
bool RefreshCollection::refreshBook (  
    const BookBaseEntry & bbe)
```

Refreshes information in database about particular book.

## Parameters

<i>bbe</i>	<a href="#">BookBaseEntry</a> object (see <a href="#">BaseKeeper::searchBook()</a> ).
------------	---

## Returns

Returns *true*, if operation has been successful.

**4.34.3.3 refreshCollection()**

```
void RefreshCollection::refreshCollection ()
```

Refreshes whole collection.

Carries out collection refreshing.

## Note

This method can throw [MLException](#) in case of errors.

**4.34.3.4 refreshFile()**

```
void RefreshCollection::refreshFile (  
    const BookBaseEntry & bbe)
```

Refreshes iformation about particular file.

## Parameters

<i>bbe</i>	<a href="#">BookBaseEntry</a> object (see <a href="#">BaseKeeper::searchBook()</a> ).
------------	---

**4.34.3.5 set\_rar\_support()**

```
void RefreshCollection::set_rar_support (  
    const bool & rar_support)
```

Enables support of rar archives.

Set **rar\_support** to *true*, if you need to parse rar archives (see also [CreateCollection::CreateCollection\(\)](#)).

## Parameters

<code>rar_support</code>	if <i>true</i> , rar archives will be parsed.
--------------------------	---

## 4.34.4 Member Data Documentation

### 4.34.4.1 bytes\_hashed

```
std::function<void(const double &hashed)> RefreshCollection::bytes_hashed
```

"Total bytes hashed" signal.

Emitted after file has been hashed, to indicate total quantity of bytes have been hashed. Bind your method to **bytes\_hashed**, if you need such information.

### 4.34.4.2 total\_bytes\_to\_hash

```
std::function<void(const double &total_hash)> RefreshCollection::total_bytes_to_hash
```

"Total bytes to hash" signal.

Emitted after files for refreshing have been collected, to indicate total quantity bytes to be hashed. Bind your method to **total\_bytes\_to\_hash**, if you need such information.

## 4.35 RemoveBook Class Reference

The [RemoveBook](#) class.

```
#include <RemoveBook.h>
```

### Public Member Functions

- [RemoveBook](#) (const std::shared\_ptr< [AuxFunc](#) > &af, const [BookBaseEntry](#) &bbe, const std::string &col\_name, const std::shared\_ptr< [BookMarks](#) > &bookmarks)  
*RemoveBook constructor.*
- void [removeBook](#) ()  
*Removes book.*

### 4.35.1 Detailed Description

The [RemoveBook](#) class.

This class contains methods to carry out book removing from collection.

## 4.35.2 Constructor & Destructor Documentation

### 4.35.2.1 RemoveBook()

```
RemoveBook::RemoveBook (
    const std::shared_ptr< AuxFunc > & af,
    const BookBaseEntry & bbe,
    const std::string & col_name,
    const std::shared_ptr< BookMarks > & bookmarks)
```

[RemoveBook](#) constructor.

## Parameters

<i>af</i>	smart pointer to <a href="#">AuxFunc</a> object.
<i>bbe</i>	<a href="#">BookBaseEntry</a> containing book info.
<i>col_name</i>	collection name.
<i>bookmarks</i>	<a href="#">BookMarks</a> object.

### 4.35.3 Member Function Documentation

#### 4.35.3.1 removeBook()

```
void RemoveBook::removeBook ()
```

Removes book.

## Note

This method can throw [MLException](#) in case of errors.

## 4.36 ReplaceTagItem Class Reference

The [ReplaceTagItem](#) class.

```
#include <ReplaceTagItem.h>
```

## Public Member Functions

- **ReplaceTagItem ()**  
*ReplaceTagItem* constructor.
- **ReplaceTagItem** (const [ReplaceTagItem](#) &other)  
*ReplaceTagItem* copy constructor.
- **ReplaceTagItem** ([ReplaceTagItem](#) &&other)  
*ReplaceTagItem* move constructor.
- **ReplaceTagItem & operator=** (const [ReplaceTagItem](#) &other)  
*operator =*
- **ReplaceTagItem & operator=** ([ReplaceTagItem](#) &&other)  
*operator =*

## Public Attributes

- std::string **tag\_to\_replace**  
*Id of tag to be replaced (see [XMLTag::tag\\_id](#)).*
- std::string **begin\_replacement**  
*Replacement for start tag element.*
- std::string **end\_replacement**  
*Replacement for end tag element.*

### 4.36.1 Detailed Description

The [ReplaceTagItem](#) class.

Auxiliary class for [FormatAnnotation](#) (see [FormatAnnotation::setTagReplacementTable\(\)](#)).

## 4.37 SelfRemovingPath Class Reference

The [SelfRemovingPath](#) class.

```
#include <SelfRemovingPath.h>
```

### Public Member Functions

- **SelfRemovingPath** ()  
*SelfRemovingPath* constructor.
- virtual **~SelfRemovingPath** ()  
*SelfRemovingPath* destructor.
- **SelfRemovingPath** (const [SelfRemovingPath](#) &other)  
*SelfRemovingPath* copy constructor.
- **SelfRemovingPath** ([SelfRemovingPath](#) &&other)  
*SelfRemovingPath* move constructor.
- [SelfRemovingPath](#) & **operator=** (const [SelfRemovingPath](#) &other)  
*operator =*
- [SelfRemovingPath](#) & **operator=** ([SelfRemovingPath](#) &&other)  
*operator =*
- [SelfRemovingPath](#) & **operator=** (const std::filesystem::path &path)  
*operator =*
- [SelfRemovingPath](#) (const std::filesystem::path &path)  
*SelfRemovingPath* constructor.

### Public Attributes

- std::filesystem::path **path**  
*Path to be removed on destruction.*

### 4.37.1 Detailed Description

The [SelfRemovingPath](#) class.

Auxiliary class. Removes underlying path on destruction, if no any copies of [SelfRemovingPath](#) object have been created. Removes path on last copy destruction otherwise.

### 4.37.2 Constructor & Destructor Documentation

#### 4.37.2.1 SelfRemovingPath()

```
SelfRemovingPath::SelfRemovingPath (
    const std::filesystem::path & path) [explicit]
```

[SelfRemovingPath](#) constructor.

## Parameters

<i>path</i>	path to be removed on destruction.
-------------	------------------------------------

### 4.37.3 Member Function Documentation

#### 4.37.3.1 operator=()

```
SelfRemovingPath & SelfRemovingPath::operator= (  
    const std::filesystem::path & path)
```

operator =

## Parameters

<i>path</i>	path to be removed on destruction.
-------------	------------------------------------

## Returns

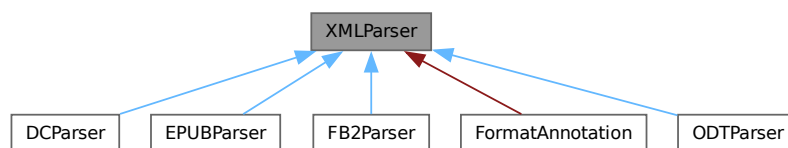
Returns self.

## 4.38 XMLParser Class Reference

The [XMLParser](#) class.

```
#include <XMLParser.h>
```

Inheritance diagram for XMLParser:



## Public Member Functions

- [XMLParser](#) (const std::shared\_ptr< [AuxFunc](#) > &af)  
*XMLParser constructor.*
- std::vector< [XMLTag](#) > [get\\_tag](#) (const std::string &book, const std::string &tag\_id)  
*Returns all tags with particular name.*
- std::string [get\\_book\\_encoding](#) (const std::string &book)  
*Returns [XML](#) document encoding.*
- std::string [get\\_element\\_attribute](#) (const std::string &element, const std::string &attr\_name)  
*Returns [XML](#) tag attribute if it was found.*
- std::vector< [XMLTag](#) > [listAllTags](#) (const std::string &book)  
*Parses [XML](#) document.*
- void [searchTag](#) (const std::vector< [XMLTag](#) > &list, const std::string &tag\_id, std::vector< [XMLTag](#) > &result)  
*Searches tag in tag list.*
- void [htmlSymbolsReplacement](#) (std::string &book)  
*Replaces symbols encoded by "&..." sequences.*
- void [removeAllTags](#) (std::string &book)  
*Removes all tag elements from [XML](#) document.*

### 4.38.1 Detailed Description

The [XMLParser](#) class.

This class contains various methods to process [XML](#) documents.

### 4.38.2 Constructor & Destructor Documentation

#### 4.38.2.1 XMLParser()

```
XMLParser::XMLParser (
    const std::shared_ptr< AuxFunc > & af)
```

[XMLParser](#) constructor.

Parameters

<i>af</i>	smart pointer to <a href="#">AuxFunc</a> constructor.
-----------	---

### 4.38.3 Member Function Documentation

#### 4.38.3.1 get\_book\_encoding()

```
std::string XMLParser::get_book_encoding (
    const std::string & book)
```

Returns [XML](#) document encoding.

Tries to get [XML](#) document header and read encoding from it.

## Parameters

<i>book</i>	<a href="#">XML</a> document content.
-------------	---------------------------------------

## Returns

Returns [XML](#) document encoding if it was found.

**4.38.3.2 get\_element\_attribute()**

```
std::string XMLParser::get_element_attribute (
    const std::string & element,
    const std::string & attr_name)
```

Returns [XML](#) tag attribute if it was found.

## Parameters

<i>element</i>	tag start element (see <a href="#">XMLTag::element</a> ).
<i>attr_name</i>	requested attribute name.

## Returns

Returns attribute content if it was found.

**4.38.3.3 get\_tag()**

```
std::vector< XMLTag > XMLParser::get_tag (
    const std::string & book,
    const std::string & tag_id)
```

Returns all tags with particular name.

This method uses [listAllTags\(\)](#) and [searchTag\(\)](#) under hood.

## Note

This method can throw [MLException](#) in case of errors.

## Parameters

<i>book</i>	<a href="#">XML</a> document content.
<i>tag↔ _id</i>	requested tag name.

## Returns

Vector of found tags.

**4.38.3.4 htmlSymbolsReplacement()**

```
void XMLParser::htmlSymbolsReplacement (
    std::string & book)
```

Replaces symbols encoded by "&..." sequences.

Replaces symbols encoded by "&..." sequences for actual values (see [this](#) for details).

## Parameters

<i>book</i>	<i>XML</i> document content symbols to be replaced in.
-------------	--

**4.38.3.5 listAllTags()**

```
std::vector< XMLTag > XMLParser::listAllTags (  
    const std::string & book)
```

Parses *XML* document.

Returns all found tags.

## Note

This method can throw *MException* in case of errors.

## Parameters

<i>book</i>	<i>XML</i> document content.
-------------	------------------------------

## Returns

Vector of found tags.

**4.38.3.6 removeAllTags()**

```
void XMLParser::removeAllTags (  
    std::string & book)
```

Removes all tag elements from *XML* document.

Simply removes all found *XML* tag elements and leaves only document content.

## Parameters

<i>book</i>	<i>XML</i> document content.
-------------	------------------------------

**4.38.3.7 searchTag()**

```
void XMLParser::searchTag (  
    const std::vector< XMLTag > & list,  
    const std::string & tag_id,  
    std::vector< XMLTag > & result)
```

Searches tag in tag list.

Searches tags in tag list by given tag name.



## Parameters

<i>list</i>	tag list search to be carried out in.
<i>tag↔ _id</i>	tag name for tags to be searched.
<i>result</i>	vector for results (can be not empty, this methods appends found tags to existing results).

## 4.39 XMLTag Class Reference

The [XMLTag](#) class.

```
#include <XMLTag.h>
```

### Public Member Functions

- **XMLTag** ()  
*XMLTag* constructor.
- **XMLTag** (const [XMLTag](#) &other)  
*XMLTag* copy constructor.
- **XMLTag & operator=** (const [XMLTag](#) &other)  
*operator =*
- **XMLTag** ([XMLTag](#) &&other)  
*XMLTag* move constructor.
- **XMLTag & operator=** ([XMLTag](#) &&other)  
*operator =*
- bool **hasContent** () const  
*Checks if tag has content.*

### Public Attributes

- std::string **element**  
*Tag start element content.*
- std::string **tag\_id**  
*Tag name.*
- std::string::size\_type **content\_start**  
*Index of first byte of tag content.*
- std::string::size\_type **content\_end**  
*Index of last byte of tag content.*
- std::vector< [XMLTag](#) > **tag\_list**  
*List of [XML](#) tags, found in tag content (if any).*

### 4.39.1 Detailed Description

The [XMLTag](#) class.

Auxiliary class for [XMLParser](#). Contains tag start element content, tag name, index of tag content first byte in [XML](#) document and index of content last byte. Also contains list of tags were found in tag content.

## 4.39.2 Member Function Documentation

### 4.39.2.1 hasContent()

```
bool XMLTag::hasContent () const
```

Checks if tag has content.

This method returns *true*, if **content\_start** and **content\_end** are not equal to `std::string::npos`.

#### Returns

*true* if tag has content.

## 4.39.3 Member Data Documentation

### 4.39.3.1 content\_end

```
std::string::size_type XMLTag::content_end
```

Index of last byte of tag content.

Index of last byte of tag content in *XML* document. Can be equal to `std::string::npos`, if tag does not have content.

### 4.39.3.2 content\_start

```
std::string::size_type XMLTag::content_start
```

Index of first byte of tag content.

Index of first byte of tag content in *XML* document. Start of tag can be found by **element** size subtraction from **content\_start**. If **content\_start** value is equal to `std::string::npos`, it indicates error on tag reading (even if tag does not have any content).

### 4.39.3.3 element

```
std::string XMLTag::element
```

Tag start element content.

Tag start element content including opening "<" and closing ">" symbols.

# Index

- add\_to\_existing\_archive
  - AddBook, [8](#)
- add\_to\_existing\_archive\_dir
  - AddBook, [8](#)
- AddBook, [7](#)
  - add\_to\_existing\_archive, [8](#)
  - add\_to\_existing\_archive\_dir, [8](#)
  - AddBook, [8](#)
  - archive\_filenames, [9](#)
  - overwrite\_archive, [9](#)
  - overwrite\_archive\_dir, [9](#)
  - simple\_add, [10](#)
  - simple\_add\_dir, [10](#)
- already\_hashed
  - CreateCollection, [46](#)
- arch\_parser
  - ARCHParser, [15](#)
- ArchEntry, [10](#)
- archive\_filenames
  - AddBook, [9](#)
- ArchiveFileEntry, [11](#)
- ArchiveRemoveEntry, [12](#)
- ARCHParser, [13](#)
  - arch\_parser, [15](#)
  - ARCHParser, [15](#)
- auth\_show\_progr
  - BaseKeeper, [27](#)
- AuxFunc, [16](#)
  - copy\_book\_callback, [17](#)
  - create, [17](#)
  - detect\_encoding, [18](#)
  - get\_activated, [18](#)
  - get\_charset\_conv\_quantity, [18](#)
  - get\_converter\_by\_number, [18](#)
  - get\_extension, [19](#)
  - get\_genre\_list, [19](#)
  - get\_selfpath, [19](#)
  - get\_supported\_archive\_types\_packing, [19](#)
  - get\_supported\_archive\_types\_unpacking, [20](#)
  - get\_supported\_types, [20](#)
  - getDJVUContext, [20](#)
  - homePath, [20](#)
  - html\_to\_utf8, [20](#)
  - if\_supported\_type, [21](#)
  - libgcrypt\_error\_handling, [21](#)
  - open\_book\_callback, [21](#)
  - randomFileName, [22](#)
  - share\_path, [22](#)
  - stringToLower, [22](#)
  - temp\_path, [22](#)
  - time\_t\_to\_date, [22](#)
  - to\_hex, [23](#)
  - to\_utf\_8, [23](#)
  - utf8\_to\_system, [23](#)
  - utf\_8\_to, [24](#)
- base64
  - BookInfoEntry, [32](#)
- base\_path
  - CreateCollection, [46](#)
- BaseKeeper, [24](#)
  - auth\_show\_progr, [27](#)
  - BaseKeeper, [25](#)
  - booksWithNotes, [25](#)
  - collectionAuthors, [25](#)
  - get\_base\_vector, [26](#)
  - get\_books\_path, [26](#)
  - loadCollection, [26](#)
  - searchBook, [27](#)
- book\_genre
  - BookParseEntry, [34](#)
- book\_name
  - BookParseEntry, [34](#)
- book\_path
  - BookParseEntry, [35](#)
- BookBaseEntry, [27](#)
  - BookBaseEntry, [28](#)
- BookInfo, [29](#)
  - BookInfo, [29](#)
  - get\_book\_info, [29](#)
  - set\_dpi, [30](#)
- BookInfoEntry, [30](#)
  - base64, [32](#)
  - cover\_types, [31](#)
  - error, [32](#)
  - file, [32](#)
  - rgb, [32](#)
  - rgba, [32](#)
- BookMarks, [32](#)
  - BookMarks, [32](#)
  - createBookMark, [33](#)
  - getBookMarks, [33](#)
  - removeBookMark, [33](#)
- BookParseEntry, [33](#)
  - book\_genre, [34](#)
  - book\_name, [34](#)
  - book\_path, [35](#)
- books
  - FileParseEntry, [62](#)

- books\_path
  - CreateCollection, 47
- booksWithNotes
  - BaseKeeper, 25
- buf\_hashing
  - Hasher, 69
- ByteOrder, 35
  - ByteOrder, 37, 38
  - get\_big, 39
  - get\_little, 39
  - get\_native, 39
  - operator=, 39–41
  - set\_big, 41
  - set\_little, 42
- bytes\_hashed
  - RefreshCollection, 102
- cancel
  - Hasher, 70
- closeBaseFile
  - CreateCollection, 45
- collectionAuthors
  - BaseKeeper, 25
- content\_end
  - XMLTag, 110
- content\_start
  - XMLTag, 110
- copy\_book\_callback
  - AuxFunc, 17
- cover\_types
  - BookInfoEntry, 31
- create
  - AuxFunc, 17
- createArchFile
  - LibArchive, 72
- createBookMark
  - BookMarks, 33
- CreateCollection, 42
  - already\_hashed, 46
  - base\_path, 46
  - books\_path, 47
  - closeBaseFile, 45
  - CreateCollection, 44, 45
  - createCollection, 45
  - current\_bytes, 47
  - need\_to\_parse, 47
  - openBaseFile, 45
  - progress, 47
  - pulse, 47
  - rar\_support, 48
  - signal\_total\_bytes, 48
  - threadRegulator, 46
  - write\_file\_to\_base, 46
- createCollection
  - CreateCollection, 45
- current\_bytes
  - CreateCollection, 47
- dcAuthor
  - DCParser, 50
- dcDate
  - DCParser, 50
- dcDescription
  - DCParser, 51
- dcGenre
  - DCParser, 51
- dcIdentifier
  - DCParser, 51
- dcLanguage
  - DCParser, 52
- DCParser, 48
  - dcAuthor, 50
  - dcDate, 50
  - dcDescription, 51
  - dcGenre, 51
  - dcIdentifier, 51
  - dcLanguage, 52
  - DCParser, 50
  - dcPublisher, 52
  - dcSource, 52
  - dcTitle, 53
- dcPublisher
  - DCParser, 52
- dcSource
  - DCParser, 52
- dcTitle
  - DCParser, 53
- detect\_encoding
  - AuxFunc, 18
- djvu\_book\_info
  - DJVUParser, 54
- djvu\_parser
  - DJVUParser, 54
- DJVUParser, 53
  - djvu\_book\_info, 54
  - djvu\_parser, 54
  - DJVUParser, 54
- editBook
  - RefreshCollection, 100
- editNote
  - NotesKeeper, 86
- ElectroBookInfoEntry, 55
- element
  - XMLTag, 110
- epub\_book\_info
  - EPUBParser, 59
- epub\_parser
  - EPUBParser, 59
- EPUBParser, 56
  - epub\_book\_info, 59
  - epub\_parser, 59
  - EPUBParser, 58
- error
  - BookInfoEntry, 32
- fb2\_book\_info
  - FB2Parser, 61

- fb2\_parser
  - FB2Parser, 61
- FB2Parser, 59
  - fb2\_book\_info, 61
  - fb2\_parser, 61
  - FB2Parser, 61
- file
  - BookInfoEntry, 32
- file\_hash
  - FileParseEntry, 62
- file\_hashing
  - Hasher, 69
- file\_rel\_path
  - FileParseEntry, 63
- fileinfo
  - LibArchive, 72
- fileNames
  - LibArchive, 73
- fileNamesStream
  - LibArchive, 73
- FileParseEntry, 62
  - books, 62
  - file\_hash, 62
  - file\_rel\_path, 63
- final\_cleaning
  - FormatAnnotation, 64
- FormatAnnotation, 63
  - final\_cleaning, 64
  - FormatAnnotation, 64
  - remove\_escape\_sequences, 65
  - removeAllTags, 65
  - replace\_tags, 65
  - setTagReplacementTable, 65
- Genre, 66
  - genre\_code, 66
  - genre\_name, 66
- genre\_code
  - Genre, 66
- genre\_name
  - Genre, 66
- GenreGroup, 67
  - group\_name, 68
- get\_activated
  - AuxFunc, 18
- get\_base\_vector
  - BaseKeeper, 26
- get\_big
  - ByteOrder, 39
- get\_book\_encoding
  - XMLParser, 107
- get\_book\_info
  - BookInfo, 29
- get\_books\_path
  - BaseKeeper, 26
- get\_charset\_conv\_quantity
  - AuxFunc, 18
- get\_converter\_by\_number
  - AuxFunc, 18
- get\_element\_attribute
  - XMLParser, 107
- get\_extension
  - AuxFunc, 19
- get\_genre\_list
  - AuxFunc, 19
- get\_little
  - ByteOrder, 39
- get\_native
  - ByteOrder, 39
- get\_selfpath
  - AuxFunc, 19
- get\_supported\_archive\_types\_packing
  - AuxFunc, 19
- get\_supported\_archive\_types\_unpacking
  - AuxFunc, 20
- get\_supported\_types
  - AuxFunc, 20
- get\_tag
  - XMLParser, 107
- getBookMarks
  - BookMarks, 33
- getDJVUContext
  - AuxFunc, 20
- getNote
  - NotesKeeper, 86
- getNotesForCollection
  - NotesKeeper, 86
- group\_name
  - GenreGroup, 68
- hasContent
  - XMLTag, 110
- Hasher, 68
  - buf\_hashing, 69
  - cancel, 70
  - file\_hashing, 69
  - Hasher, 69
  - stop\_all\_signal, 70
- homePath
  - AuxFunc, 20
- html\_to\_utf8
  - AuxFunc, 20
- htmlSymbolsReplacement
  - XMLParser, 107
- if\_supported\_type
  - AuxFunc, 21
- LibArchive, 70
  - createArchFile, 72
  - fileinfo, 72
  - fileNames, 73
  - fileNamesStream, 73
  - LibArchive, 72
  - libarchive\_error, 74
  - libarchive\_packing, 74
  - libarchive\_read\_entry, 75
  - libarchive\_read\_entry\_str, 75

- libarchive\_read\_init, 75
- libarchive\_read\_init\_fallback, 76
- libarchive\_remove\_entry, 76
- libarchive\_remove\_init, 76
- libarchive\_write\_data, 77
- libarchive\_write\_data\_from\_file, 77
- libarchive\_write\_directory, 77
- libarchive\_write\_file, 78
- libarchive\_write\_init, 78
- unpackByFileNameStream, 80
- unpackByFileNameStreamStr, 80
- unpackByPosition, 80
- unpackByPositionStr, 81
- writeToArchive, 81
- libarchive\_error
  - LibArchive, 74
- libarchive\_packing
  - LibArchive, 74
- libarchive\_read\_entry
  - LibArchive, 75
- libarchive\_read\_entry\_str
  - LibArchive, 75
- libarchive\_read\_init
  - LibArchive, 75
- libarchive\_read\_init\_fallback
  - LibArchive, 76
- libarchive\_remove\_entry
  - LibArchive, 76
- libarchive\_remove\_init
  - LibArchive, 76
- libarchive\_write\_data
  - LibArchive, 77
- libarchive\_write\_data\_from\_file
  - LibArchive, 77
- libarchive\_write\_directory
  - LibArchive, 77
- libarchive\_write\_file
  - LibArchive, 78
- libarchive\_write\_init
  - LibArchive, 78
- libgcrypt\_error\_handling
  - AuxFunc, 21
- listAllTags
  - XMLParser, 108
- loadBase
  - NotesKeeper, 86
- loadCollection
  - BaseKeeper, 26
- MLBookProc, 1
- MLException, 82
  - MLException, 83
  - what, 83
- need\_to\_parse
  - CreateCollection, 47
- NotesBaseEntry, 83
  - NotesBaseEntry, 84
- NotesKeeper, 84
- editNote, 86
- getNote, 86
- getNotesForCollection, 86
- loadBase, 86
- NotesKeeper, 85
- readNote, 87
- readNoteText, 87
- refreshCollection, 87
- removeCollection, 88
- removeNotes, 88
- odtBookInfo
  - ODTParser, 92
- ODTParser, 89
  - odtBookInfo, 92
  - ODTParser, 91
  - odtParser, 92
- odtParser
  - ODTParser, 92
- OmpLockGuard, 92
  - OmpLockGuard, 93
- open\_book
  - OpenBook, 94
- open\_book\_callback
  - AuxFunc, 21
- openBaseFile
  - CreateCollection, 45
- OpenBook, 93
  - open\_book, 94
  - OpenBook, 94
- operator=
  - ByteOrder, 39–41
  - SelfRemovingPath, 105
- overwrite\_archive
  - AddBook, 9
- overwrite\_archive\_dir
  - AddBook, 9
- PaperBookInfoEntry, 95
- pdf\_annotation\_n\_cover
  - PDFParser, 96
- pdf\_parser
  - PDFParser, 96
- PDFParser, 95
  - pdf\_annotation\_n\_cover, 96
  - pdf\_parser, 96
  - PDFParser, 96
- progress
  - CreateCollection, 47
- pulse
  - CreateCollection, 47
- randomFileName
  - AuxFunc, 22
- rar\_support
  - CreateCollection, 48
- readNote
  - NotesKeeper, 87
- readNoteText

- NotesKeeper, 87
- refreshBook
  - RefreshCollection, 101
- RefreshCollection, 97
  - bytes\_hashed, 102
  - editBook, 100
  - refreshBook, 101
  - RefreshCollection, 100
  - refreshCollection, 101
  - refreshFile, 101
  - set\_rar\_support, 101
  - total\_bytes\_to\_hash, 102
- refreshCollection
  - NotesKeeper, 87
  - RefreshCollection, 101
- refreshFile
  - RefreshCollection, 101
- remove\_escape\_sequences
  - FormatAnnotation, 65
- removeAllTags
  - FormatAnnotation, 65
  - XMLParser, 108
- RemoveBook, 102
  - RemoveBook, 102
  - removeBook, 103
- removeBook
  - RemoveBook, 103
- removeBookMark
  - BookMarks, 33
- removeCollection
  - NotesKeeper, 88
- removeNotes
  - NotesKeeper, 88
- replace\_tags
  - FormatAnnotation, 65
- ReplaceTagItem, 103
- rgb
  - BookInfoEntry, 32
- rgba
  - BookInfoEntry, 32
- searchBook
  - BaseKeeper, 27
- searchTag
  - XMLParser, 108
- SelfRemovingPath, 104
  - operator=, 105
  - SelfRemovingPath, 104
- set\_big
  - ByteOrder, 41
- set\_dpi
  - BookInfo, 30
- set\_little
  - ByteOrder, 42
- set\_rar\_support
  - RefreshCollection, 101
- setTagReplacementTable
  - FormatAnnotation, 65
- share\_path
  - AuxFunc, 22
- signal\_total\_bytes
  - CreateCollection, 48
- simple\_add
  - AddBook, 10
- simple\_add\_dir
  - AddBook, 10
- stop\_all\_signal
  - Hasher, 70
- stringToLower
  - AuxFunc, 22
- temp\_path
  - AuxFunc, 22
- threadRegulator
  - CreateCollection, 46
- time\_t\_to\_date
  - AuxFunc, 22
- to\_hex
  - AuxFunc, 23
- to\_utf\_8
  - AuxFunc, 23
- total\_bytes\_to\_hash
  - RefreshCollection, 102
- unpackByFileNameStream
  - LibArchive, 80
- unpackByFileNameStreamStr
  - LibArchive, 80
- unpackByPosition
  - LibArchive, 80
- unpackByPositionStr
  - LibArchive, 81
- utf8\_to\_system
  - AuxFunc, 23
- utf\_8\_to
  - AuxFunc, 24
- what
  - MLEException, 83
- write\_file\_to\_base
  - CreateCollection, 46
- writeToArchive
  - LibArchive, 81
- XMLParser, 105
  - get\_book\_encoding, 106
  - get\_element\_attribute, 107
  - get\_tag, 107
  - htmlSymbolsReplacement, 107
  - listAllTags, 108
  - removeAllTags, 108
  - searchTag, 108
  - XMLParser, 106
- XMLTag, 109
  - content\_end, 110
  - content\_start, 110
  - element, 110
  - hasContent, 110