mdict4j Documentation

Hiroshi Miura

2023 年 11 月 09 日

目次

第 1 章 Translations 3

第 2 章 Development status 5

第 3 章 License 7

第 4 章 Contents 9

第 5 章 Index 31

MDict is one of popular dictionary formats. Mdict4j is a parser library for mdict dictionary.

第 1 章

Translations

Chinese: https://mdict4j.readthedocs.io/zh_CN/latest Japanese: https://mdict4j.readthedocs.io/ja/latest English: https://mdict4j.readthedocs.io/en/latest

第 2 章

Development status

A status of library development is considered as *Alpha*.

第 3 章

License

Mdict4j is distributed under GNU General Public License version 3 or (at your option) any later version. Please see LICENSE file in project root.

第 4 章

Contents

When you want to use the library in your application project, please check "How to use" section. When you are interested in joining the mdict4j project, "File format specification" section is good start point.

4.1 mdict4j の利用法

4.1.1 インストール

Gradle をビルドシステムとして利用している場合には、mavenCentral から mdict4j をインストールできます。

```
dependencies {
    implementation 'tokyo.northside:mdict4j:0.4.0'
}
```

We changed groupId from version 0.4.0. If you want to use old versions, please use group id 'io.github.eb4j'

4.1.2 辞書ファイルのロード

辞書ファイルをロードして MDictDictionary オブジェクトを作成し、このオブジェクトから辞書検索できます。

```
Path dictionaryPath = Paths.get("foo.mdx");
MDictDictionary dictionary = MDictDictionary.loadDictionary(dictionaryPath);
```

MDictDictionary オブジェクトには、MDX ファイルの属性を調べるメソッドがあります。

```
if (dictionary.isMDX()) {
    System.out.println("loaded file is .mdx");
}
```

(次のページに続く)

(前のページからの続き)

```
if (StandardCharsets.UTF_8.equals(dictionary.getEncoding())) {
    System.out.println("MDX file encoding is UTF-8");
}
if (dictionary.isHeaderEncrypted()) {
    System.out.println("MDX file is encrypted.");
}
if (dictionary.isIndexEncrypted()) {
    System.out.println("MDX index part is encrypted.");
}
System.out.printf("MDX version: %d, format: %s", dictionary.getMdxVersion(), dictionary.
↪getFormat());
System.out.println(dictionary.getCreationDate());
System.out.println(dictionary.getTitle());
System.out.println(dictionary.getDescription());
```

You can invoke query from MDictDictionary object. MDictDictionary::readArticles method returns a list of entries. MDictDictionary::readArticles method returns exact match results. MDictDictionary::readArticlesPredictive method returns predictive search match results.

```
for (Map.Entry<String, String> entry: dictionary.readArticles("hello")) {
    System.out.println("<div><span>%s</span>: %s</div>", entry.getKey(), entry.
↪getValue());
}
```

You will see something like `hello:` **簌好**

```
for (Map.Entry<String, String> entry: dictionary.readArticlesPredictive("happ")) {
    System.out.println("<div><span>%s</span>: %s</div>", entry.getKey(), entry.
↪getValue());
}
```

You will see something like `happy:` **快簌** `happiness:` **幸福**

MDD データファイルをよみこむときは MDictDictionary##loadDictionaryData メソッドを使用します。

```
Path dataPath = Paths.get("foo.mdx");
MDictDictionary dictData = MDictDictionary.loadDictionaryData(dataPath);
if (!dictData.isMDX()) {
    System.out.println("loaded file is .mdd");
}
```

(次のページに続く)

(前のページからの続き)

```
Map.Entry<String, Object> entry = dictData.getEntries("/audio/test.mp3").get(0);
Object value = entry.getValue();
byte[] buf = dictData.getData((Long) value);  // buf contains mp3 data.
Tika tika = new Tika();
String mediaType = tika.detect(buf);
System.out.println("Media type should be audio/mpeg: %s", mediaType);
```

Please check javadoc for details.

```
./gradlew javadoc
```

4.2 Security Policy

4.2.1 Supported Versions

Version	Status
0.5.x	Development version
< 0.5	not supported

4.2.2 Reporting a Vulnerability

Please disclose security vulnerabilities privately at miurahr@linux.com in English or Japanese, no Chinese.

4.3 Contribution guide

This is contribution guide for mdict4j project. You are welcome to send a Pull-Request, reporting bugs and ask questions.

4.3.1 Resources

- Project owner: Hiroshi Miura
- Bug Tracker: CodeBerg issue Tracker
- Status: Alpha
- Activity: moderate

4.3.2 Bug triage

Every report to github issue tracker should be in triage. whether it is bug, question or invalid.

4.3.3 Send patch

Here is small amount rule when you want to send patch the project;

1. every proposal for modification should send as 'Pull Request'
1. each pull request can consist of multiple commits.
1. you are encourage to split modifications to individual commits that are logical subpart.

4.3.4 CI tests

The project configured to use Azure Pipelines for regression test. You can see test results on badge and see details in a web page linked from badge.

4.4 Contributor Covenant Code of Conduct

4.4.1 Our Pledge

We as members, contributors, and leaders pledge to make participation in our community a harassment-free experience for everyone, regardless of age, body size, visible or invisible disability, ethnicity, sex characteristics, gender identity and expression, level of experience, education, socio-economic status, nationality, personal appearance, race, religion, or sexual identity and orientation.

We pledge to act and interact in ways that contribute to an open, welcoming, diverse, inclusive, and healthy community.

4.4.2 Our Standards

Examples of behavior that contributes to a positive environment for our community include:

- Demonstrating empathy and kindness toward other people
- Being respectful of differing opinions, viewpoints, and experiences
- Giving and gracefully accepting constructive feedback
- Accepting responsibility and apologizing to those affected by our mistakes, and learning from the experience

- Focusing on what is best not just for us as individuals, but for the overall community

Examples of unacceptable behavior include:

- The use of sexualized language or imagery, and sexual attention or advances of any kind
- Trolling, insulting or derogatory comments, and personal or political attacks
- Public or private harassment
- Publishing others' private information, such as a physical or email address, without their explicit permission
- Other conduct which could reasonably be considered inappropriate in a professional setting

4.4.3 Enforcement Responsibilities

Community leaders are responsible for clarifying and enforcing our standards of acceptable behavior and will take appropriate and fair corrective action in response to any behavior that they deem inappropriate, threatening, offensive, or harmful.

Community leaders have the right and responsibility to remove, edit, or reject comments, commits, code, wiki edits, issues, and other contributions that are not aligned to this Code of Conduct, and will communicate reasons for moderation decisions when appropriate.

4.4.4 Scope

This Code of Conduct applies within all community spaces, and also applies when an individual is officially representing the community in public spaces. Examples of representing our community include using an official e-mail address, posting via an official social media account, or acting as an appointed representative at an online or offline event.

4.4.5 Enforcement

Instances of abusive, harassing, or otherwise unacceptable behavior may be reported to the community leaders responsible for enforcement at Hiroshi Miura <miurahr@linux.com>. All complaints will be reviewed and investigated promptly and fairly.

All community leaders are obligated to respect the privacy and security of the reporter of any incident.

4.4.6 Enforcement Guidelines

Community leaders will follow these Community Impact Guidelines in determining the consequences for any action they deem in violation of this Code of Conduct:

Phase 1. Correction

Community Impact:

Use of inappropriate language or other behavior deemed unprofessional or unwelcome in the community.

Consequence: A

private, written warning from community leaders, providing clarity around the nature of the violation and an explanation of why the behavior was inappropriate. A public apology may be requested.

Phase 2. Warning

Community Impact: A

violation through a single incident or series of actions.

Consequence:

A warning with consequences for continued behavior. No interaction with the people involved, including unsolicited interaction with those enforcing the Code of Conduct, for a specified period of time. This includes avoiding interactions in community spaces as well as external channels like social media. Violating these terms may lead to a temporary or permanent ban.

Phase 3. Temporary Ban

Community Impact: A

serious violation of community standards, including sustained inappropriate behavior.

Consequence: A

temporary ban from any sort of interaction or public communication with the community for a specified period of time. No public or private interaction with the people involved, including unsolicited interaction with those enforcing the Code of Conduct, is allowed during this period. Violating these terms may lead to a permanent ban.

Phase 4. Permanent Ban

Community Impact:

Demonstrating a pattern of violation of community standards, including sustained inappropriate behavior, harassment of an individual, or aggression toward or disparagement of classes of individuals.

Consequence: A

permanent ban from any sort of public interaction within the community.

4.4.7 Attribution

This Code of Conduct is adapted from the Contributor Covenant version 2.0. You can take it from Contributor Covenant homepage.

Community Impact Guidelines were inspired by Mozilla's code of conduct enforcement ladder.

For answers to common questions about this code of conduct, see the FAQ or its translations.

4.5 File format specification

4.5.1 Reference origin and license

A fileformat.md here is the reference description from writemdict project with copyright by zhansliu, distributed under the MIT License, the term is attached as MIT.txt.

4.5.2 Introduction

This is a description of version 2.0 of the MDX and MDD file format, used by the MDict dictionary software. The software is not open-source, nor is the file format openly specified, so the following description is based on reverse-engineering, and is likely incomplete and inaccurate in its details.

Most of the information comes from https://bitbucket.org/xwang/mdict-analysis. While xwang mostly focuses on being able to read this unknown format, I have added details that are necessary to also write MDX files.

4.5.3 Concepts

MDX and MDD files are both designed to store an associative array of pairs (keyword, record).

For MDX files, the information stored is typically a dictionary. The keyword and record are both (Unicode) strings, with the keyword being the headword for the dictionary entry, and the record giving a description of that word. An example of an MDX entry could be:

- keyword: "reverse engineering"
- record: "noun: a process of analyzing and studying an object or device, in order to understand its inner workings"

MDD files are instead designed to store binary data. Typically, the keyword is a file path, and the record is the contents of that file. As an example, we may have:

- keyword: "\image.png"
- record: 0x89 0x50 0x4e 0x47 0x0d 0x0a 0x1a 0x0a...

MDX files is designed to store a dictionary, i.e. a collection of pairs (keyword, record), which could be, for example, keyword="reverse engineering", record="noun: a process of analyzing and studying an object or device, in order to understand its inner workings".

Typically, MDD files are associated with an MDX file of the same name (but with extension .mdx instead of .mdd), and contains resources to be included in the text of MDX files. For example, and entry of the MDX file might contain the HTML code `<img src="/images/image.png" />`, in which case the MDict software will look for the entry "\image.png" in the MDD file.

4.5.4 File structure

The basic file structure is a follows:

MDX File	
`header_sect`	Header section. See "Header Section" below.
`keyword_sect`	Keyword section. See "Keyword Section" below.
`record_sect`	Record section. See "Record Section" below.

4.5.5 Header Section

header_sect	Length	
length	4 bytes	Length of header_str, in bytes. Big-endian.
header_str	varying	An XML string, encoded in UTF-16LE. See below for details.
checksum	4 bytes	ADLER32 checksum of header_str, stored little-endian.

The header_str consists of a single, XML tag dictionary, with various attributes. For MDX files, they look like this: (newlines added for clarity)

```
<Dictionary
GeneratedByEngineVersion="2.0"
RequiredEngineVersion="2.0"
Encrypted="2"
Encoding="UTF8"
Format="Html"
CreationDate="2015-01-01"
Compact="No"
Compat="No"
KeyCaseSensitive="No"
Description="This is a <i>test dictionary</i>."
Title="My dictionary"
DataSourceFormat="106"
StyleSheet=""
RegisterBy="Email"
RegCode="0102030405060708090A0B0C0D0E0F"/>
```

For MDD files, we have instead:

```
<Library_Data
GeneratedByEngineVersion="2.0"
RequiredEngineVersion="2.0"
Encrypted="2"
Format=""
CreationDate="2015-01-01"
Compact="No"
Compat="No"
KeyCaseSensitive="No"
```

(次のページに続く)

(前のページからの続き)

```
Description="This is a <i>test dictionary</i>."
Title="My dictionary"
DataSourceFormat="106"
StyleSheet=""
RegisterBy="Email"
RegCode="0102030405060708090A0B0C0D0E0F"/>
```

The meaning of the attributes are explained below:

Attribute	Description
Generated	The version of the file format. This document describes version 2.0. Apart from this, version 1.2 is also possible.
RequiredE	Presumably the lowest format version compatible with this version.
Encrypted	An integer between 0 and 3 (inclusive). If the lower bit is set, indicates that the first part of the keyword section is encrypted, as described in the section *Keyword header encryption*. If the upper bit is set, indicates that the keyword index is encrypted, using the scheme described in *Keyword index encryption*.
Encoding	Only used for MDX files. The encoding used for text in the document. Possible values are "UTF-8", "UTF-16" (uses little-endian encoding), "GBK", and "Big5". For MDD files, the encoding used for the keywords (file paths) is always UTF-16, and the records consist of binary data.
Format	The format of the dictionary entry texts. Possible values include "Html" and "Text". For MDD files, this must be empty.
CreationD	The date the dictionary was created.
Compact	If this is "Yes", indicates the dictionary entries is in an Mdict-specific compact format, where certain string are replaced according to the scheme specified in StyleSheet. See the documentation for the official MdxBuilder client for details.
Compat	Appears to be a typo for Compact, which certain versions of the official Mdict client look for instead of Compact.
KeyCaseSe	Indicates to the dictionary reader whether or not keys should be treated in a case-insensitive manner.
Descripti	A description of the dictionary, which appears as the ":about" page in the official MDict client.
Title	The title of the dictionary.
DataSourc	Unknown.
StyleShee	Used in conjunction with the Compact option. See the documentation for the official MdxBuilder client for details.
RegisterB	Either "EMail" or "DeviceID". Only used if the lower bit of Encrypted is set. Indicates which piece of user-identifying data is used to encrypt the encryption key. See the section *Keyword header encryption* for details.
RegCode	When keyword header encryption is used (see *Keyword header encryption*), this is one way to deliver the encrypted key. In this case, this is a string consisting of 32 hexadecimal digits.

4.5.6 Keyword Section

The keyword section contains all the keywords in the dictionary, divided into blocks, as well as information about the sizes of these blocks.

keyword_sect	Lengt	
num_blocks	8 bytes	Number of items in key_blocks. Big-endian. Possibly encrypted, see below.
num_entries	8 bytes	Total number of keywords. Big-endian. Possibly encrypted, see below.
key_index_decom	8 bytes	Number of bytes in decompressed version of key_index. Big-endian. Possibly encrypted, see below.
key_index_comp_	8 bytes	Number of bytes in compressed version of key_index (including the comp_type and checksum parts). Big-endian. Possibly encrypted, see below.
key_blocks_len	8 bytes	Total number of bytes taken up by key_blocks. Big-endian. Possibly encrypted, see below.
checksum	4 bytes	ADLER32 checksum of the preceding 40 bytes. If those are encrypted, it is the checksum of the decrypted version. Big-endian.
key_index	varying	The keyword index, compressed and possibly encrypted. See below.
key_blocks[0]	varying	A compressed block containing keywords, compressed. See below.
...	...	...
key_blocks[num_	varying	...

4.5.7 Keyword header encryption:

If the parameter Encrypted in the header has the lowest bit set (i.e. Encrypted | 1 is nonzero), then the 40-byte block from num_blocks are encrypted. The encryption used is Salsa20/8 (Salsa20 with 8 rounds instead of 20). In pseudo-Python:

```
def encrypt(message, key):
    salsa20_8_init(key_length = 128, #128 bits
       iv_length = 64, # 64 bits
       ivs = b"\0\0\0\0\0\0\0\0"), #64 bits of zeros)
    return salsa20_8_encrypt(message, key)
```

(次のページに続く)

(前のページからの続き)

```
encrypted_block = encrypt(unencrypted_block, key=ripemd128(encryption_key))
```

Here, `encryption_key` is the dictionary password specified on creation of the dictionary.

This `encryption_key` is not distributed directly. Instead it is further encrypted, using a piece of data, `user_id`, that is specific to the user or the client machine, according to the following scheme:

```
reg_code = encrypt(ripemd128(encryption_key), ripemd128(user_id))
```

The string `user_id` can be either an email address ("example@example.com") that the user enters into his/her MDict client, or a device ID ("12345678-90AB-CDEF-0123-4567890A") which the MDict client obtains in different ways depending on the platform. The choice of which one to use depends on the attribute `RegisterBy` in the file header. (See *Header section*.) In either case, `user_id` is an ASCII-encoded string. On certain platforms, the official MDict client seems to default to the DeviceID being the empty string.

The 128-bit `reg_code` is then distributed to the user. This can be done in two ways:

- If the MDX file is called `dictionary.mdx`, the dictionary reader should look for a file called `dictionary.key` in the same directory, which contains `reg_code` as a 32-digit hexadecimal string.
- Otherwise, `reg_code` can be included in the header of the MDX file, as the attribute `RegCode`.

Keyword index

The keyword index lists some basic data about the key blocks. It is compressed (see "Compression"), and possibly encrypted (see "Keyword index encryption"). After decompression and decryption, it looks like this:

decompress(key_	Leng	
num_entries[0]	8 bytes	Number of keywords in the first keyword block.
first_size[0]	2 bytes	Length of first_word[0], not including trailing null character. In number of "basic units" for the encoding, so e.g. bytes for UTF-8, and 2-byte units for UTF-16.
first_word[0]	varying	The first keyword (alphabetically) in the key_blocks[0] keyword block. Encoding given by Encoding attribute in the header.
last_size[0]	2 bytes	Length of last_word[0], not including trailing null character. In number of "basic units" for the encoding, so e.g. bytes for UTF-8, and 2-byte units for UTF-16.
last_word[0]	varying	The last keyword (alphabetically) in the key_blocks[0] keyword block. Encoding given by Encoding attribute in the header.
comp_size[0]	8 bytes	Compressed size of key_blocks[0].
decomp_size[0]	8 bytes	Decompressed size of key_blocks[0].
num_entries[1]	8 bytes	...
...	...	...
decomp_size[nur	8 bytes	...

Keyword index encryption:

If the parameter Encrypted in the header has its second-lowest bit set (i.e. Encrypted | 2 is nonzero), then the keyword index is further encrypted. In this case, the comp_type and checksum fields will be unchanged (refer to the section Compression), the following C function will be used to encrypt the compressed_data part, after compression.

```
#define SWAPNIBBLE(byte) (((byte)>>4) | ((byte)<<4))
void encrypt(unsigned char* buf, size_t buflen, unsigned char* key, size_t keylen) {
        unsigned char prev=0x36;
        for(size_t i=0; i < buflen; i++) {
                buf[i] = SWAPNIBBLE(buf[i] ^ ((unsigned char)i) ^ key[i%keylen] ^␣
↪previous);
                previous = buf[i];
        }
}
```

The encryption key used is ripemd128(checksum + "\x95\x36\x00\x00"), where + denotes string concatenation.

Keyword blocks

Each keyword is compressed (see "Compression"). After decompressing, they look like this:

decompress(key_blocks[(	Length	
offset[0]	8 bytes	Offset where the record corresponding to key[0] can be found, see below. Big-endian.
key[0]	varying	The first keyword in the dictionary, null-terminated and encoded using Encoding.
offset[1]	8 bytes	...
key[1]	varying	...
...	...	...

The offset should be interpreted as follows: Decompress all record blocks, and concatenate them together, and let records denote the resulting array of bytes. The record corresponding to key[i] then starts at records[offset[i]].

4.5.8 Record section

The record section looks like this:

record_sect	Lengt	
num_blocks	8 bytes	Number items in record_blocks. Does not need to equal the number of keyword blocks. Big-endian.
num_entries	8 bytes	Total number of records in dictionary. Should be equal to keyword_sect.num_entries. Big-endian.
index_len	8 bytes	Total size of the comp_size[i] and decomp_size[i] variables, in bytes. In other words, should equal 16 times num_blocks. Big-endian.
blocks_len	8 bytes	Total size of the rec_block[i] sections, in bytes. Big-endian.
comp_size[0]	8 bytes	Length of rec_block[0], in bytes. Big-endian.
decomp_size[0]	8 bytes	Decompressed size of rec_block[i], in bytes. Big-endian.
comp_size[1]	8 bytes	Length of rec_block[1], in bytes. Big-endian.
...	...	...
decomp_size[num_	8 bytes	...
rec_block[0]	varying	A compressed block containing records. See below.
...	...	...
rec_block[num_bl	varying	...

Record block

Each record block is compressed (see "Compression"). After decompressing, they look like this:

decompress(rec_block[0	Length	
record[0]	varying	The first record. If in an MDX file, this is null-terminated and encoded using Encoding.
record[1]	varying	...
...	...	...

4.5.9 Compression:

Various data blocks are compressed using the same scheme. These all look like these:

compress(data)	Length	
comp_type	4 bytes	Compression type. See below.
checksum	4 bytes	ADLER32 checksum of the uncompressed data. Big-endian.
compressed_data	varying	Compressed version of data.

The compression type can be indicated by comp_type. There are three options:

- If comp_type is '\x00\x00\x00\x00', then no compression is applied at all, and compressed_data is equal to data.
- If comp_type is '\x01\x00\x00\x00', LZO compression is used.
- If comp_type is '\x02\x00\x00\x00', zlib compression is used. It so happens that the zlib compression format appends an ADLER32 checksum, so in this case, checksum will be equal to the last four bytes of compressed_data.

4.6 Change Log

All notable changes to this project will be documented in this file.

4.6.1 Unreleased

4.6.2 0.5.4

- Gradle: use version catalog
- Bump Jackson@2.14.2
- Bump bouncycastle@1.76
- Bump Gradle@8.3

4.6.3 0.5.3

- Gradle: add spotless config
- Update javadoc descriptions

4.6.4 0.5.2

- Fixed to return only a target article text (#89)
- Fix loading error degraded in V0.5.0 for V2/UTF-16 dictionary

4.6.5 0.5.0

- Bump required JAVA 11
- Introduce module-info.java
- Fix the case when dictionary is Ver1.2/UTF-16
- Bump Gradle@7.6
- Bump versions:
  - Gradle plugin tokyo.northside.sphinx@1.0.4
  - jetbrains annotations@23.1.0
  - jackson@2.14.0
  - groovy-all@3.0.14
  - junit@5.9.0
  - jsoup@1.15.3
  - tika@2.6.0
  - slf4j-simple@2.0.5
- Replace cache with SimpleLRUCache
  - Drop Caffeine dependency
- Remove python files in docs/_extensions

4.6.6 0.4.3

- Fix issue in FatJar
- Change dependencies versions
  - Tika@2.4.1 (Fix CVE-2022-33879)
  - Jackson@2.12.7 (Fix FatJar issue)
- Update test case

4.6.7 0.4.2

- Docs: Add .readthedocs.yaml document build configuration file
- Docs: Fix language configuration
- Docs: Update dependency MyST-Parser to support Markdown
- Docs: Fix changelog links

4.6.8 0.4.0

- Change group id to "tokyo.northside"

4.6.9 0.3.1

- Fix public method name typo
  - change main load method name MDictDictionary#loadDictionary

4.6.10 0.3.0

- Caching index with Caffeine
  - mdict4j automatically cache queried articles, maximum 1000 entries in 15 min.
- Don't automatically index in lower case
  - User need to check MDictDictionary#isKeyCaseSensitive() whether query lower case or not.
- Bump versions
  - slf4j-simple@1.7.36

  - spotless@6.2.2

- Fix and add test cases

4.6.11 0.2.4

- Introduce readArticles and readArticlesPredictive method
- Introduce readData method
- Change behavior
  - Do not search again with lower case.
- Bump versions
  - Tika@2.3.0
  - Gradle git-version@0.13.0
  - Spotless@6.2.1
  - SpotBugs@5.0.5
  - Actions setup-java@2.5.0
  - Actions gradle-build-action@v2

4.6.12 0.2.3

- Bump Gradle/gradle-build-action@v2

4.6.13 0.2.2

- Bump versions
  - Jackson@2.13.1
  - JUnit@5.8.2
  - Gradle kotlin@1.6.10
  - actions setup-java@2.4.0
  - BouncyCastle@1.70
  - Gradle@7.3.2

4.6.14 0.2.1

- Support dictionary that use UTF-16(LE) as encoding.
  - Force endian to LE when UTF-16 is specified even lacking BOM.

4.6.15 0.2.0

- Support MDD file loading
- Test: Apache Tika for dependency
- Improve test

4.6.16 0.1.4

- Bump jackson@2.10.5
- Experimental implementation for .MDD file
- Update and fix v1 parser
- Update and fix dictionary key loading
- Improve tests

4.6.17 0.1.3

- Change jackson version to 2.7.4.

4.6.18 0.1.2

- Fix publish configurations

4.6.19 0.1.1

- First release

4.6.20 0.1.0

- First internal release

第 5 章

Index

- genindex
- modindex
- search