mobi2epub Documentation

Release 0.01

JA

January 16, 2013

CONTENTS

Contents:

CHAPTER

ONE

MOBI::

1.1 mobi::st_c_section

class mobi::**st_c_section**

Container used for simpler management of Content array by passing it's size along with it

1.1.1 st_c_section variables

public

uint8* st_c_section::**content**

unsigned int st_c_section::**size**

1.1.2 st_c_section methods

public

st_c_section()

st_c_section(unsigned int *x*)

sets up st_c_section::content to uint8[x]

and st_c_section::size to x

1.2 mobi::mobireader

class mobi::**mobireader**

Opens .mobi files and reads as much data as possible.

```
std::string  foo = "foo.mobi";

mobi::mobireader bar(foo);

std::cout << bar.get_title();
```

1.2.1 mobireader variables

private

st_palmdoc_db mobireader::**db_header**

st_palmdoc mobireader::**pd_header**

st_mobi mobireader::**mobi_header**

vector mobireader::**section_offsets**
byte offset of every data section in file

st_c_section mobireader::**c_section**

string mobireader::**input_file_name**
name of the file, because ifstream is too cool to keep that.

ifstream* mobireader::**file**
All it should do is get passed to the handler

compression* mobireader::**reader**
Pointer to new instance of mobi::compression

header_handler* mobireader::**handler**
pointer to new instance of header_handler

char* mobireader::**title**
title of the book set by set_title() and set_default_title() methods

1.2.2 mobireader methods

private

mobireader::void **parse_header**()
loads up headers structures and section vector

if mobireader::db_header type doesn't equal BOOKMOBI, throws mobi::invalid_file_exception

mobireader::void **set_compression**()
decides which one of the mobi::compression classes should *reader point to

throws unsupported_compressiontype_exception for dictionary compression. mainly because i haven't found it in any of my books.

std::string mobireader::**get_section_uncompressed**(unsigned int *s*) const
Handles uncompressing and returning section from a valid range of section_offsets vector

header_out_of_range_exception

public:

mobireader::**mobireader**(std::string& *input_file_name*)

mobireader::**mobireader**(const mobireader& *m*)

mobireader::**mobireader**()

mobireader::**~mobireader**()

mobireader::void **set_default_title**()
reads the default title from file

char* mobireader::**set_title**(const char* *s*) const

char* mobireader::**get_title**() const
returns current book title.

std::string mobireader::**get_html**() const
iterates *reader over sections and returns html-like text.

std::string mobireader::**get_file_name**() const
returns file name from mobireader::input_file_name

void mobireader::**load_file**(std::string& *input_file_name*)
loads file from string and sets up handler and file pointers.

void mobireader::**operator=**(const mobireader& *m*)
obviously copies mobireader class onto another instance. calls load_file() to check if the file is still ok.

1.3 mobi::header_handler

class header_handler
Handles reading data from the .mobi files, so the mobi::mobireader methods can be a bit cleaner.

1.3.1 header_handler variables

private

1.3.2 header_handler methods

std::ifstream* header_handler::**file**

public:

header_handler(std::ifstream* *file*)
Sets up file pointer to mobireader::file

header_handler& header_handler::**offset**(unsigned int *offset*)
Sets offset in file returns reference to self

header_handler& header_handler::**skip**(unsigned int *skip*)
Skips n bytes in file returns reference to self

header_handler& header_handler::**read**(type& *i*)
Reads sizeof i from file and saves it to i

returns reference to header_handler

header_handler& header_handler::**read**(type& *i*, unsigned int *n*)
Reads n times sizeof i from file and saves it to i

Should be overloaded for every necessary data type

or just use templates, which i won't implement, because there's already too much syntax noise as it is, without the chevron brackets.

returns reference to self

1.4 mobi::compression

class mobi::**compression**
abstract class, providing base for pd_compression, no_compression and hd_compression

1.4.1 compression variables

private

std::string **destination**

1.4.2 compression methods

public:

void compression::**uncompress**(uint8_t* *src*, size_t *srcLen*) = 0

Note: pure virtual function

Should be overloaded with:

```
if(!this->destination.empty())
    this->destination.clear();
(decompression algorithm that appends stuff to this->destination)
return;
```

std::string compression::**output_raw**()

Note: virtual function

Usually should just:

```
return this->destination;
```

class mobi::**pd_compression**
inherits from compression

Uses PalmDoc algorithm:

1. Read a byte from the compressed stream. If the byte is
2. 0x00: "1 literal" copy that byte unmodified to the decompressed stream.
3. 0x09 to 0x7f: "1 literal" copy that byte unmodified to the decompressed stream.
4. 0x01 to 0x08: "literals": the byte is interpreted as a count from 1 to 8, and that many literals are copied unmodified from the compressed stream to the decompressed stream.

5. 0x80 to 0xbf: "length, distance" pair: the 2 leftmost bits of this byte ('10') are discarded, and the following 6 bits are combined with the 8 bits of the next byte to make a 14 bit "distance, length" item. Those 14 bits are broken into 11 bits of distance backwards from the current location in the uncompressed text, and 3 bits of length to copy from that point (copying n+3 bytes, 3 to 10 bytes).
6. 0xc0 to 0xff: "byte pair": this byte is decoded into 2 characters: a space character, and a letter formed from this byte XORed with 0x80.
7. Repeat from the beginning until there is no more bytes in the compressed f

Except it doesn't copy 0x00. Because who in the right mind would want to do that?

throws `mobi::not_palmdoc_compression_exception`

from check at stage 5.

class `mobi::no_compression`
inherits from `compression`

It's just:

```
if(!this->destination.empty())
    this->destination.clear();
this->destination.append((char *)src, srcLen);
```

in `compression::uncompress()`

class `mobi::hd_compression`
inherits from `compression`

currently not implemented, because none of the books i've found use it.

1.5 separate mobi::functions

`mobi::strcmp_is_a_worthless_pos` (char* *x*, char* *y*, int* *n*)
compares x and y arrays up to n. returns true if they match

void **bswap** (uint16_t& *x*)

void **bswap** (uint32_t& *x*)

void **bswap** (uint64_t& *x*)
switch variable endianess on little endian architecture.

void **unretardify_header** (st_palmdoc_db& *x*)

void **unretardify_header** (st_palmdoc& *x*)

void **unretardify_header** (st_mobi& *x*)
bswap every value in those structures.

type st_palmdoc_db
Should be equal to 78 bytes.

```
char   name[32];
uint16_t flags;
uint16_t version;
uint32_t c_time;
uint32_t m_time;
uint32_t b_time;
uint32_t mod_num;
uint32_t app_info;
```

```
uint32_t sort_info;
char   type[8]; //BOOKMOBI
uint32_t u_id_seed;
uint32_t next_record_list;
uint16_t num_records;
```

type st_palmdoc

Should be equal to 16 bytes.

```
uint16_t compression
uint16_t garbage
uint32_t text_length
uint16_t record_count
uint16_t record_size
uint32_t current_pos
```

type st_mobi

Should be equal to 232 bytes.

```
char   id[4]
uint32_t header_len
uint32_t mobi_type
uint32_t text_encoding
uint32_t u_id
uint32_t file_version
uint32_t ortographic_index
uint32_t inflection_index
uint32_t index_names
uint32_t index_keys
uint32_t extra_index0
uint32_t extra_index1
uint32_t extra_index2
uint32_t extra_index3
uint32_t extra_index4
uint32_t extra_index5
uint32_t first_nonbook_index
uint32_t full_name_offset
uint32_t full_name_length
uint32_t locale
uint32_t input_language
uint32_t output_language
uint32_t min_version
uint32_t first_image_index
uint32_t huffman_record_offset
uint32_t huffman_record_count
uint32_t huffman_table_offset
uint32_t huffman_table_length
uint32_t exth_flags
char   garbage[32]
uint32_t drm_offset
uint32_t drm_count
uint32_t drm_size
uint32_t drm_flags
char   garbage2[12]
uint16_t first_content_record_number
uint16_t last_content_record_number
uint32_t unknown
uint32_t fcis_record_number
uint32_t fcis_record_count
```

```
uint32_t flis_record_number
uint32_t flis_record_count
uint64_t unk0
uint32_t unk1
uint32_t unk2
uint32_t unk3
uint32_t unk4
uint32_t extra_record_data_flags
uint32_t indx_record_offset
```

CHAPTER

TWO

EPUB::

It's a namespace for classes reading epub, creating epub and converting TO epub

std::string **container_xml**
: http://wiki.mobileread.com/wiki/EPUB#container.xml

```
<?xml version=\"1.0\"?>\n\
<container version=\"1.0\" xmlns=\"urn:oasis:names:tc:opendocument:xmlns:container\">\n\
<rootfiles>\n\
    <rootfile full-path=\"OEBPS/content.opf\" media-type=\"application/oebps-package+xml\"/>\n\
    \n\
</rootfiles>\n\
</container>"
```

std::string **mimetype**
: http://wiki.mobileread.com/wiki/EPUB#mimetype

```
"application/epub+zip";
```

std::string **content_opf**
: http://wiki.mobileread.com/wiki/EPUB#OPF

```
<?xml version=\"1.0\" encoding=\"utf-8\" standalone=\"yes\"?>\n\
<package xmlns=\"http://www.idpf.org/2007/opf\" unique-identifier=\"BookId\" version=\"2.0\">\n\
    <metadata xmlns:dc=\"http://purl.org/dc/elements/1.1/\" xmlns:opf=\"http://www.idpf.org/2007
        <dc:title>%1%</dc:title> \n\
    </metadata>\n\
    <manifest>\n\n\
\
%2%\n\   //itemid
\
    </manifest>\n\
    <spine toc=\"ncx\">\n\n\
\
%3%\n\ //itemref
\
    </spine>\n\
</package>\n\
```

title string comes from `mobireader::get_title()`

std::string **itemid**
: format string for `content_opf`

std::string **itemref**
: format string for `content_opf`

2.1 epub::mobi2epub

class epub**::mobi2epub**
Class converting .mobi files to .epub

and because of lame .zip support writes them on the disk immediately.

2.1.1 mobi2epub variables

private

boost::filesystem::path **path_out**
Path to output file *obviously including file name*

Used in mobi2epub::directory_to_epub()

boost::filesystem::path **path_tmp**
Path to temporary directory used to save the entire uncompressed file structures

and because everybody should use tmpfs, everything happens in the RAM anyway

mobi::mobireader **m**
instance of mobi::mobireader copied in mobi2epub::mobi2epub()

mutable bool **safe**
Tells if it's ok to run mobi2epub::directory_structure()

mutable bool **no_cleanup**
Makes mobi2epub::cleanup() exit immediately

mutable bool **vanilla_out**
checks if mobi2epub::set_out() was called and path_out comes from the user

2.1.2 mobi2epub methods

private

void mobi2epub::**directory_structure**() const
Creates the static part of every .epub file:

```
path_tmp/
         OEBPS/
         OEBPS/text/
         OEBPS/META-INF/
         META-INF/container.xml
         mimetype
```

checks if already exists, if so ask to whether remove or set to safe and continue

void mobi2epub::**cleanup**() const

> **Warning:** Removes recursively every file in path_tmp without asking if safe is set to **True** should be handled with caution.

no_cleanup decides whether it should return instantly

throws epub::user_wants_out_exception if the user decides to quit

void mobi2epub::**gen_content_opf**(std::stringstream& *itemids*, std::stringstream& *itemrefs*) const
generates content_opf and saves it to:

```
OEBPS/content.opf
```

void mobi2epub::**operator=**(mobi2epub& *m*)
makes the class uncopyable

mobi2epub::**mobi2epub**(mobi2epub& *m*)
makes the class uncopyable even more

void mobi2epub::**set_out**(std::string& *s*)
sets vanilla_out to **False**, then changes path_tmp to

public

mobi2epub::**mobi2epub**(const mobi::mobireader& *m*, bool *safe=false*, bool *no_cleanup=false*)

void mobi2epub::**save_to_directory**() const
calls mobi2epub::directory_structure() and mobireader::get_html()

splits the html on every new chapter mark:

```
<mbp:pagebreak/>
```

then iterates over it to

1. sanitize it as xhtml with epub::html::tidyhtml
2. save it to path_tmp /OEBPS/text
3. build up itemid and itemref lists, which get passed to mobi2epub::gen_content_opf() later on

void mobi2epub::**save_to_directory**(std::string *s*)
calls mobi2epub::set_out() and then proceeds to mobi2epub::save_to_directory()

void mobi2epub::**directory_to_epub**() const
calls zip executable to compress everything in path_tmp and save it to path_out

---

Note: if it somehow compiles on windows and nothing fails up to this point, should throw epub::terrible_operating_system_exception

---

void mobi2epub::**directory_to_epub**(std::string *s*)
calls mobi2epub::set_out() before calling actual mobi2epub::directory_to_epub()

CHAPTER

THREE

EXCEPTIONS

class mobi::**not_palmdoc_compression_exception**

class epub::**file_write_exception**

class epub::**terrible_operating_system_exception**

class epub::**zip_exit_status_exception**

class epub::**path_changed_exception**

class epub::**user_wants_out_exception**

class mobi::**no_such_file_exception**

class mobi::**invalid_file_exception**

class mobi::**invalid_compression_type_exception**

class mobi::**header_out_of_range_exception**

class mobi::**unsupported_compressiontype_exception**

CHAPTER

FOUR

INDICES AND TABLES

- *genindex*
- *modindex*
- *search*

INDEX

P

S

U

V