ECM2 specifications by Daniel Carrasco

Rev-2, August 2021

Alpha Version

Changelog

Revision	Date	Modifications
R01	2021-07	First version of the document
R02	2021-08	Revised document to add the new features

Table of content

Document

Document Objectives

This document defines the ECM2 file format and how the ecm-tools-reloaded program works, and is divided into the following sections:

section 1. This section which describes the document itself.

section 2. History

section 3. How it works

section 4. Headers specification

section 5. Data compression

section 6. Audio compression

section 7. Error Detection Code

History

The Error Code Modeler format works by removing the unnecessary data from the CD Sectors. This data is mostly the Error Detection Code and the Error Correction Code, that is main reason why the format is called ECM.

The original ECM version removes all the ECM data so is good enough and it helped me to save a lot of space on my disks. The way it works is by detecting the Mode 2 XA sectors and removing the EDC and ECC data from it, and treating the rest of the data as raw bytes (including part of the Mode 2 XA sectors).

I wanted to improve the program by removing even more data, like for example the sync, the address and redundant sub-header data. Also I wanted to create a seekable file by processing the input file block by block and placing the index in the file header. This allows to know the exact position of every sector in file by just reading the header, which opens the window to the possibility of create a plugin for PCSX to read it directly.

The first program modification works with the above in mind and reduces the file size of the resulting ecm file up to 8%. This version already creates seekable files, and just reading the header you will be able to create an index of the file to be able to seek to the desired sector very fast. Also I have moved the functions to a class to allow to reuse it into another programs.

The second program modification adds some compression methods to it, allowing to no depend of external tools. Also unlike most of the external programs, will allow to set the compression method of every stream, allowing to compress the data using LZMA and the CDDA. Also I wanted to keep the ability to seek into the file. I have modified the headers to add this functionality, and improve the readability by creating fixed size blocks.

How it works

The ECM format removes data from CD-ROM sectors depending of which kind of sector contains the source, and the following sectors are compatible with this tool:

- CDDA
- MODE1
- MODE2
- MODE2 XA1
- MODE2 XA2

The sector size is CD-ROM is fixed and contains 2352 bytes.

CDDA Sector

The CDDA sector is entirely composed by data bytes, so nothing can be removed and thus the sector size is not reduced.

2352 data bytes

There also exists a variant of this sector type which is filled by zeros. This sector variant can be easily generated and then can be safely fully removed.

Mode 1 Sector

The mode 1 sector contains 12 sync bytes, 4 header bytes, 2048 data bytes, 4 EDC bytes, 8 blank bytes and 276 ECC bytes.

Sync: 12b	Header: 4b	Data: 2048b	EDC: 4b	Blank: 8b	ECC: 276b

The ECM program will be able to keep only the data and remove the rest of the data, which can be easily generated later to restore the original sector. This will reduce the sector size by about 13%.

There also exists a variant of this sector type which its data is filled by zeros. This sector variant will be fully removed as it can be fully generated without problem.

Mode 2 Sector

The mode 2 sector is not widely used, but is also processed. This sector contains 12 sync bytes, 4 header bytes and 2336 data bytes.

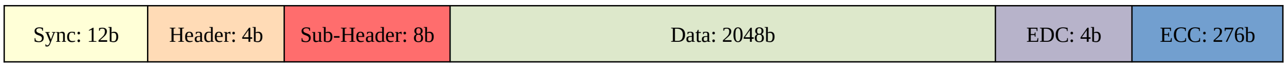
Sync: 12b	Header: 4b	Data: 2336b

The ECM program will be able to keep only the data part of the sector and remove the rest, which can be easily be generated later to restore the original sector. This will reduce the sector size by about 0.7% (not too much).

There also exists a variant of this sector type which its data is filled by zeros. This sector variant can be easily generated and then can be safely fully removed.

Mode 2 XA 1 Sector

This is the first Mode 2 Extended sector type and is very similar to Mode 1 sector. The 8 bytes blank sector is moved between the header and the data and is used as sub-header. This sector contains 12 sync bytes, 4 header bytes, 8 sub-header bytes, 2048 data bytes, 4 EDC bytes and 276 ECC bytes.

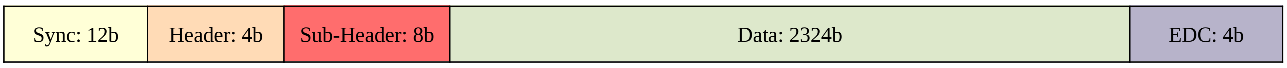


In this sector mode the data is required and also the sub-header. As the sub-header is redundant (4 bytes repeated 2 times), only 4 bytes are required. The ECM program will be able to remove the rest of data and then reduce the size about 13%. The removed data also can be generated easily later to restore the original sector.

There also exists a variant of this sector type which its data is filled by zeros. This sector variant can be mostly easily generated and then almost all can be removed. Only 4 sub-header bytes are required.

Mode 2 XA 1 Sector

This mode removes the ECC bytes and uses its space to store data, so it contains 12 sync bytes, 4 header bytes, 8 sub-header bytes, 2324 data bytes and 4 EDC bytes.



Like the XA 1 sector mode, it only requires the data bytes and 4 sub-header bytes, and the rest can be generated. This allow to reduce the size about 1.2%.

There also exists a variant of this sector type which its data is filled by zeros. This sector variant can be mostly easily generated and then almost all can be removed. Only 4 sub-header bytes are required.

Why to use ECM2

The main reason to use ECM2 is the size reduction, which in some situations is about the 13% of the original file size and can reach higher reductions if GAP sectors are found. For example, The Final Fantasy VII CD1 image has reached a reduction of about the 21% without compression, so it can be implemented into an emulator and then reduce your collection size with a much lower CPU overhead. The format also allows some compression methods which will help to reduce the file size even more.

The removed data can be generated later and recover the original sector data, so this tool will produce a lossless reduction method, which can be complemented with another compression methods like zlib, lzma, lz4 and even flac for CDDA tracks.

Headers specifications

In this section we will see the headers specifications starting by the main header, and continuing by the streams header and the sectors headers.

Main Header

The main header starts at the beginning of the file and is composed by 6 bytes. Three of them are fixed, the 4th defines the file version and the other two defines the optimization options used at encoding.

Position	**Size**	**Value**	**Description**
0x00	3 bytes	ECM	ECM container header
0x03	1 byte	2	Fourth byte in file. This byte indicates the file version which is: 0. Original ECM version developed by Neill Corlett 1. First modified developed by Daniel Carrasco which modify the way the sectors are detected, and removes more data. Is incompatible with older versions. 2. Second modification made by Daniel Carrasco which adds data and audio compression (currently in development). Is incompatible with older versions.
0x04	1 byte		1 byte containing the sectors per block info (max 255)
0x05	1 byte		1 byte containing the optimization used to generate this ECM file. Usefull for the decoding process.

Streams Header

The streams header is located right after the main header, starting at the seventh byte. First byte is the number of streams that the file contains:

Position	**Size**	**Value**	**Description**
0x00	8 bytes		Streams count in uint32_t
0x08	8 bytes		Streams header size. This was introduced to allow zlib compression in headers to reduce its size, otherwise the size can be calculated.

The equivalent struct will be:

```
struct sec_str_size {
    uint32_t count;
    uint32_t size;
};
```

Those 16 bytes will be followed by groups of 5 bytes per stream containing their info:

Position	Size	Value	Description
0x00	1 byte		Stream info: • 0: Stream format → 0 = Audio, 1 = Data. • 1 - 3: Stream compression. 1 Zlib 2 LZMA 3 LZ4 4 FLAC 5 APE 6 WAVPACK • 4 - 7: Reserved
0x01	4 bytes		The number of sectors that contains the stream
0x05	4 bytes		The end position in output file

The equivalent struct will be:

```
#pragma pack(push, 1)
struct stream {
    uint8_t type: 1;
    uint8_t compression: 3;
    uint32_t end_sector;
    uint32_t out_end_position;
};
#pragma pack(pop)
```

We set the pack to 1 to avoid the compile alignment in this struct, which will create an 8 bytes structs instead the 5 bytes that we want.

This header is intended to contains the different streams types in file, with their end position in the file and compression type. This will allow to have different compression types, for example, Zlib for data streams and WavPack for audio streams, and also will help to calculate how to decompress the input stream.

Sectors Header

The sectors header is located at the end of streams header, and the structure is very similar to this. First byte contains the sectors entries number in header:

Position	Size	Value	Description
0x00	8 bytes		Sectors count in uint32_t
0x04	8 bytes		Sectors header size. This was introduced to allow zlib compression in headers to reduce its size, otherwise the size can’t be calculated.

The equivalent struct will be:

```
struct sec_str_size {
    uint32_t count;
    uint32_t size;
};
```

Those 16 bytes will be followed by groups of 5 bytes per sector type containing the sector info:

Position	Size	Value	Description
0x00	1 byte		Sector info: • 0 - 3: Sector type. 1 CDDA 2 CDDA GAP 3 Mode 1 4 Mode 1 GAP 5 Mode 2 6 Mode 2 GAP 7 Mode 2 XA1 8 Mode 2 XA1 GAP 9 Mode 2 XA2 10 Mode 2 XA2 GAP • 4 – 7: Reserved
0x01	4 bytes		The number of sectors of that type processed. This number is equivalent to an uint32_t variable.

In this case the equivalent struct will be:

```
#pragma pack(push, 1)
struct sector {
    uint8_t mode: 4;
    uint32_t sector_count;
};
#pragma pack(pop)
```

We set the pack to 1 to avoid the alignment in the struct, which will create an 8 bytes struct instead the 5 bytes we want.

This header is intended to contains the different sectors types and number of sectors of this type in the source file. This info will be used to recover the original sector state. The entire header is compressed using zlib to save some space.

Data Stream

The data stream contains all the data copied from the source file sectors. The uncompressed data doesn’t contains any checkpoint or similar, but you will be able to seek into it just reading the sectors header. Compressed streams will require of checkpoints to sync the sectors position and be able to seek.

Data compression

The program allows to compress the data stream, using the zlib, lzma, lz4 and flac (only audio) compression libraries. This allow to have an even smaller file without use any external compression.

Zlib

Enabling the zlib compression method, the program will compress every data stream using the zlib library. This will reduce the output file size. Zlib is a good balance between speed and compression ratio.

Optionally, a set of “checkpoints” can be created into the stream with the zlib flush option Z_FULL_FLUSH. This will allow to seek into the file with tools that requires random access to sectors, like for example an emulator plugin. This option will not have any impact in normal decompression but will reduce the compression ratio, so is recommended to keep it disable if ECM file will not be used for random access. A maximum of 255 sectors per block is allowed and the number must be stored in the 5th byte of the main header to help the tool used for random access to know how many sectors exists in every block.

LZMA

Enabling the lzma compression method, the program will compress every data stream using the xz library. This will reduce the output size. Lzma compress at higher ratios than zlib compression method, but is slower in compression/decompression.

LZ4

The lz4 compression method compress less than the zlib and lzma methods, but compression/decompression is much faster, allowing to seek into the file in less time. This is very usefull when the resources in the system are limited or you want a fast seek time for an emulator.

Audio compression

The program allows to compress the audio stream, using the zlib and lzma compression libraries. This allow to have an even smaller file without use any external compression. The compression ratio of zlib and lzma libraries is lower in audio streams.

Zlib

Same as data Zlib compression.

LZMA

Same as data LZMA compression.

LZ4

Same as data LZ4 compression.

FLAC

FLAC compression is available only as audio compression method because FLAC is an audio format. Is able to also compress data sectors but only if they are untouched, and that is not usefull in ECM data compression.

Error Detection Code

At the end of the file, there are 4 bytes that correspond to the Error detection Code. This Error Detection Code is exactly the same used into the CD-ROM sectors, and adds a way to check if the extracted stream is exact to the input file.

Position	Size	Value	Description
Last 4 bytes	4 bytes		Error detection code calculated using the same method using in the sectors

Maybe in the future I'll add MD5 or SHA hashing methods, but this will change the ECM2 format.