LZ4 Frame Format

Notices

Version

1.5.0

Introduction

The purpose of this document is to define a lossless compressed data format, that is independent of CPU type, operating system, file system and character set, suitable for File compression, Pipe and streaming compression using the LZ4 algorithm : http://code.google.com/p/lz4/

The data can be produced or consumed, even for an arbitrarily long sequentially presented input data stream, using only an a priori bounded amount of intermediate storage, and hence can be used in data communications. The format uses the LZ4 compression method, and optional xxHash-32 checksum method, for detection of data corruption.

The data format defined by this specification does not attempt to allow random access to compressed data.

This specification is intended for use by implementers of software to compress data into LZ4 format and/or decompress data from LZ4 format. The text of the specification assumes a basic background in programming at the level of bits and other primitive data representations.

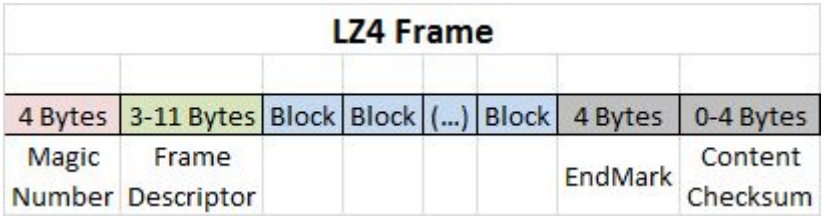
Unless otherwise indicated below, a compliant compressor must produce data sets that conform to the specifications presented here. It doesn't need to support all options though.

A compliant decompressor must be able to decompress at least one working set of parameters that conforms to the specifications presented here. It may also ignore checksums. Whenever it does not support a specific parameter used within the compressed stream, it must produce a non-ambiguous error code and associated error message explaining which parameter is unsupported.

Distribution of this document is unlimited.

Summary :

General Structure of LZ4 Framing format



Magic Number

4 Bytes, Little endian format.
Value : **0x184D2204**

Frame Descriptor

3 to 11 Bytes, to be detailed in the next part.
Most important part of the spec.

Data Blocks

To be detailed later on.
That's where compressed data is stored.

EndMark

The flow of blocks ends when the last data block has a size of "**0**".
The size is expressed as a 32-bits value.

Content Checksum

Content Checksum verify that the full content has been decoded correctly.
The content checksum is the result of xxh32() hash function digesting the original (decoded) data as input, and a seed of zero.
Content checksum is only present when its associated flag is set in the framing descriptor. Content Checksum validates the result, that all blocks were fully transmitted in the correct order and without error, and also that the encoding/decoding process itself generated no distortion. Its usage is recommended.

Frame Concatenation

In some circumstances, it may be preferable to append multiple frames, for example in order to add new data to an existing compressed file without re-framing it.

In such case, each frame has its own set of descriptor flags. Each frame is considered independent. The only relation between frames is their sequential order.

The ability to decode multiple concatenated frames within a single stream or file is left outside of this specification. As an example, the reference lz4 command line utility behavior is to decode all concatenated frames in their sequential order.

Frame Descriptor

Frame Descriptor			
FLG	**BD**	*0-8 Bytes*	**HC**
Descriptor Flags		*Content Size*	Descriptor Checksum

Descriptor Flags																
FLG								**BD**								
7	6	5	4	3	2	1	0		7	6	5	4	3	2	1	0
Version Number		Block Independence	Block Checksum	Content Size	Content Checksum	*Reserved*			*Reserved*		Block Maximum Size		*Reserved*			

The descriptor uses a minimum of 3 bytes, and up to 11 bytes depending on optional parameters.
In the picture, bit 7 is highest bit, while bit 0 is lowest.

Version Number :

2-bits field, **must** be set to "**01**".
Any other value cannot be decoded by this version of the specification.
Other version numbers will use different flag layouts.

Block Independence flag :

If this flag is set to "1", blocks are independent, and can therefore be decoded independently, in parallel.
If this flag is set to "0", each block depends on previous ones for decoding (up to LZ4 window size, which is 64 KB). In this case, it's necessary to decode all blocks in sequence.

Block dependency improves compression ratio, especially for small blocks. On the other hand, it makes jumps or multi-threaded decoding impossible.

Block checksum flag :

If this flag is set, each data block will be followed by a 4-bytes checksum, calculated by using the xxHash-32 algorithm on the raw (compressed) data block.
The intention is to detect data corruption (storage or transmission errors) immediately, before decoding.
Block checksum usage is optional.

Content Size flag :

If this flag is set, the original (uncompressed) size of data included within the frame will be present as an 8 bytes unsigned value, little endian format, after the flags.

Recommended value : "**0**" (not present)

Content checksum flag :

If this flag is set, a content checksum will be appended after the EoS mark.

Recommended value : "**1**" (content checksum is present)

Block Maximum Size :

This information is intended to help the decoder allocate the right amount of memory.
Size here refers to the original (uncompressed) data size.
Block Maximum Size is one value among the following table :

Block Maximum Sizes							
0	1	2	3	4	5	6	7
N/A	N/A	N/A	N/A	64 KB	256 KB	1 MB	4 MB

The decoder may refuse to allocate block sizes above a (system-specific) size.
Unused values may be used in a future revision of the spec.
A decoder conformant to the current version of the spec is only able to decode blocksizes defined in this spec.

Reserved bits :

Value of reserved bits **must** be **0** (zero).
Reserved bit might be used in a future version of the specification, to enable any (yet-to-decide) optional feature.
If this happens, a decoder respecting the current version of the specification shall not be able to decode such a frame.

Content Size

This is the original (uncompressed) size.
This information is optional, and only present if the associated flag is set.
Content size is provided using unsigned 8 Bytes, for a maximum of 16 HexaBytes.

Format is Little endian.
This field has no impact on decoding, it just informs the decoder how much data the frame holds (for example, to display it during decoding process, or for verification purpose). It can be safely skipped by a conformant decoder.

Header Checksum :

One-byte checksum of all descriptor fields, including optional ones when present.
The byte is second byte of xxh32() : { (xxh32()>>8) & 0xFF } ,
using zero as a seed,
and the full Frame Descriptor as an input (*including* optional fields when they are present).
A different checksum indicates an error in the descriptor.

Data Blocks

Data Blocks		
4 Bytes	data	*0-4 Bytes*
Block Size	Compressed data	*Block checksum*

Block Size

This field uses **4-bytes,** format is little-endian.

The highest bit is "**1**" if data in the block is uncompressed.

The highest bit is "**0**" if data in the block is compressed by LZ4.

All other bits give the size, in bytes, of the following data block (the size does not include the checksum if present).

Block Size shall never be larger than Block Maximum Size. Such a thing could happen when the original data is incompressible. In this case, such a data block shall be passed in uncompressed format.

Data

Where the actual data to decode stands. It might be compressed or not, depending on previous field indications.
Uncompressed size of Data can be any size, up to "block maximum size".
Note that data block is not necessarily full : an arbitrary "flush" may happen anytime. Any block can be "partially filled".

Block checksum :

Only present if the associated flag is set.
This is a 4-bytes checksum value, in little endian format,
calculated by using the xxHash-32 algorithm on the raw (undecoded) data block,
and a seed of zero.
The intention is to detect data corruption (storage or transmission errors) *before* decoding.

Block checksum is cumulative with Content checksum.

Skippable Frames

Skippable Frame		
4 Bytes	4 Bytes	User Data
Magic Number	Frame Size	

Skippable frames allow the integration of user-defined data into a flow of concatenated frames.
Its design is pretty straightforward, with the sole objective to allow the decoder to quickly skip over user-defined data and continue decoding.

For the purpose of facilitating identification, it is discouraged to start a flow of concatenated frames with a skippable frame. If there is a need to start such a flow with some user data encapsulated into a skippable frame, it's recommended to start will a zero-byte LZ4 frame followed by a skippable frame. This will make it easier for file type identifiers.

Magic Number

4 Bytes, Little endian format.
Value : **`0x184D2A5X`**, which means any value from **`0x184D2A50 to 0x184D2A5F.`** All 16 values are valid to identify a skippable frame.

Frame Size

This is the size, in bytes, of the following User Data (without including the magic number nor the size field itself).
4 Bytes, Little endian format, unsigned 32-bits.
This means User Data can't be bigger than (2^32-1) Bytes.

User Data

User Data can be anything. Data will just be skipped by the decoder.

Legacy frame

Legacy Format						
4 Bytes	4 Bytes	data	4 Bytes	data	x N times....	\EOF
Magic Number	Compressed Size	Compressed data	Compressed Size	Compressed data		End of File marker

The Legacy frame format was defined into the initial versions of “LZ4Demo”.
Newer compressors should not use this format anymore, since it is too restrictive.
It is recommended that decompressors shall be able to decode this format during the transition period.

Main properties of legacy format :
- Fixed block size : 8 MB.
- All blocks must be completely filled, except the last one.
- All blocks are always compressed, even when compression is detrimental.
- The last block is detected either because it is followed by the “EOF” (End of File) mark, or because it is followed by a known Frame Magic Number.
- No checksum
- Convention is Little endian

Magic Number

4 Bytes, Little endian format.
Value : **`0x184C2102`**

Block Compressed Size

This is the size, in bytes, of the following compressed data block.
4 Bytes, Little endian format.

Data

Where the actual data stands.
Data is always compressed, even when compression is detrimental (i.e. larger than original size).

Appendix

Version changes

1.5 : removed Dictionary ID from specification

1.4.1 : changed wording from “stream” to “frame”

1.4 : added skippable streams, re-added stream checksum

1.3 : modified header checksum

1.2 : reduced choice of “block size”, to postpone decision on “dynamic size of BlockSize Field”.

1.1 : optional fields are now part of the descriptor

1.0 : changed “block size” specification, adding a compressed/uncompressed flag

0.9 : reduced scale of “block maximum size” table

0.8 : removed : high compression flag

0.7 : removed : stream checksum

0.6 : settled : stream size uses 8 bytes, endian convention is little endian

0.5: added copyright notice

0.4 : changed format to Google Doc compatible OpenDocument