# FFC File Format Specification

Szymon Grabowski, Tomasz Kowalski and Robert Susik

2025-11-09

## Contents

**1 FFC File Format Specification v1.0** **2**
1.1 Description . . . 2
1.2 File Structure Overview . . . 2
1.3 Header . . . 2
1.4 Blocks . . . 2
1.4.1 Block metadata . . . 3
1.4.2 Block content . . . 3
1.5 Compression statistics . . . 4
1.6 Subblock meta format . . . 4
1.7 File layout . . . 4
1.8 Decompression . . . 5
1.8.1 Handling EOLs . . . 5
1.8.2 Handling case flags . . . 5

# 1 FFC File Format Specification v1.0

## 1.1 Description

This section describes the binary structure of files compressed with the Fast FASTA Compressor (FFC). Use this as a reference to implement compressor/decompressor. All integers use little-endian byte ordering.

## 1.2 File Structure Overview

An ffc archive file consists of the following sections (in order):

1. **Header** (variable size)
2. **Blocks** (variable size), including one terminator block
3. **Compression statistics** (fixed size)

## 1.3 Header

The file starts with a header of length at least 56 bytes.

| Offset | Size (bytes) | Type | Field name | Description |
|---|---|---|---|---|
| 0 | 8 | int64 | `ffc_header` | Magic number (0x6366662e) |
| 8 | 4 | uint32 | `version` | Version (0xMMNNPPPP: major.minor.patch) |
| 12 | 4 | int32 | `chunk_size` | Chunk size |
| 16 | 4 | int32 | `max_block_size` | Max block size |
| 20 | 4 | int32 | `case_compression_flag` | Case stream compression flag |
| 24 | 4 | int32 | `raw_compression_flag` | RAW stream compression flag |
| 28 | 4 | int32 | `dna_compression_flag` | DNA stream compression flag |
| 32 | 4 | int32 | `mix_compression_flag` | MIX stream compression flag |
| 36 | 4 | int32 | `subblocks_meta_compression_flag` | Subblocks meta stream compression flag |
| 40 | 4 | uint32 | `crc` | CRC32 of the original file |
| 44 | 8 | int64 | `orig_file_timestamp` | Original file timestamp (UNIX time) |
| 52 | 4 | int32 | `orig_filename_length` | Original filename length |
| 56 | `orig_filename_length` | text | `orig_filename` | Original filename (optional) |

`chunk_size` must be a multiple of 8. If `chunk_size` is non-zero, then the size of each subblock of type DNA, NNN, MIX must be a multiple of `chunk_size`. Value 0 means that each DNA subblock size must be a multiple of 8, but other subblock sizes are then arbitrary.

`max_block_size` is limited by $2^{30} - 1$ bytes.

The `*_compression_flag` fields store information about the applied compression mode for a particular stream type. The value of -1 is the default (adaptive mode), 0 means no compression, values [1, 22] correspond to zstd compression modes (larger values: generally better compression and slower). These fields, stored only for statistical and debugging purposes, should be ignored by the decompressor, because the format allows changing the compression method for a particular stream from block to block (e.g., in the adaptive mode).

The default values of the fields CRC32 of the original file, Original file timestamp (UNIX time) and Original filename length are 0s. If the value is 0, then a decompressor should ignore the respective field.

Original filename is of size `orig_filename_length`. Note it does not contain a terminator character.

## 1.4 Blocks

For each block in the archive, in their natural order, its metadata followed by block content are stored.

### 1.4.1 Block metadata

The metadata for each block takes 64 bytes.

| Offset | Size (bytes) | Type | Field name | Description |
|---|---|---|---|---|
| 0 | 8 | int64 | `block_start` | Block start (offset in original file) |
| 8 | 4 | int32 | `block_size` | Block size (in original file) |
| 12 | 4 | uint32 | `block_compressed_size` | Block compressed length |
| 16 | 4 | int32 | `case_mask_compressed_size` | Case mask compressed size |
| 20 | 4 | int32 | `raw_stream_size` | Raw stream size |
| 24 | 4 | int32 | `raw_stream_compressed_size` | Raw stream compressed size |
| 28 | 4 | int32 | `dna_stream_size` | DNA stream size |
| 32 | 4 | int32 | `dna_stream_compressed_size` | DNA stream compressed size |
| 36 | 4 | int32 | `mix_stream_size` | MIX stream size |
| 40 | 4 | int32 | `mix_stream_compressed_size` | MIX stream compressed size |
| 44 | 4 | int32 | `subblocks_count` | Subblocks count |
| 48 | 4 | int32 | `subblocks_meta_compressed_size` | Subblocks meta compressed size |
| 52 | 4 | int32 | `first_EOL_offset` | First EOL offset |
| 56 | 4 | int32 | `seq_line_length` | Line length in sequences |
| 60 | 4 | int32 | `seq_headers_count` | Sequence headers count |

If all fields in a block metadata are 0s, then this block is considered to be the last one (TERMINATOR BLOCK).

### 1.4.2 Block content

For each block, the following compressed streams are stored in order (sizes are given in the corresponding fields in block metadata).

Each stream starts with a 1-byte header indicating the compression method:

- `0` (`NO_CODER`): uncompressed
- `7` (`ZSTD_CODER`): compressed with Zstandard.

The operation of stripping this header and (optional) zstd decompression is called "decoding" in this section.

1. **Case mask stream** (`case_mask_compressed_size` bytes before decoding)
   - Packed case flags (1 bit per symbol).
   - Represents upper/lower case for each symbol in the block.
   - Data are packed into octets of bytes (64 flags per octet), using a special layout.
   - The size of this stream, when decoded, is ⌊ (block_size + 63) / 64 ⌋ * 8 bytes.
2. **RAW stream** (`raw_stream_compressed_size` bytes before decoding)
   - Contains arbitrary symbols (byte values).
   - Data are not packed.
   - Its size, after decoding, does not have to be a multiple of `chunk_size`.
3. **DNA stream** (`dna_stream_compressed_size` bytes before decoding)
   - Contains only A, C, G, T symbols.
   - Data are packed as 1-byte chunks containing 4 symbols (2 bits per symbol: `A=0`, `C=1`, `T=2`, `G=3`).
   - The order of packed symbols in a byte is: the 1st DNA symbol - bits 0 and 1 (i.e., two least significant bits), the 2nd
   - DNA symbol - bits 2 and 3, the 3rd DNA symbol - bits 4 and 5, the 4th DNA symbol - bits 6 and 7.
   - Its size, after decoding, has to be a multiple of `chunk_size` / 4 if `chunk_size` is positive, or a multiple of 2 if `chunk_size` is 0.
4. **MIX stream** (`mix_stream_compressed_size` bytes before decoding)
   - Contains arbitrary symbols (byte values).
   - Data are not packed.
   - Its size, after decoding, has to be a multiple of `chunk_size` if `chunk_size` is positive, or is arbitrary if `chunk_size` is 0.
5. **Subblocks meta stream** (`subblocks_meta_compressed_size` bytes before decoding)
   - Subblocks meta stream contains `subblocks_count` uint32 entries, where each entry represents a subblock metadata (see Subblock meta format section).
   - The size of this stream, when decoded, is `subblocks_count` * 4 bytes.

## 1.5 Compression statistics

At the end of the archive, a statistics structure of size 32 bytes is written.

| Offset | Size (bytes) | Type | Field name | Description |
|---|---|---|---|---|
| 0 | 8 | int64 | `blocks_count` | Blocks count |
| 8 | 8 | int64 | `original_file_size` | Original file size |
| 16 | 8 | int64 | `number_of_sequences` | Number of sequences |
| 24 | 8 | int64 | `streams_size` | Streams size |

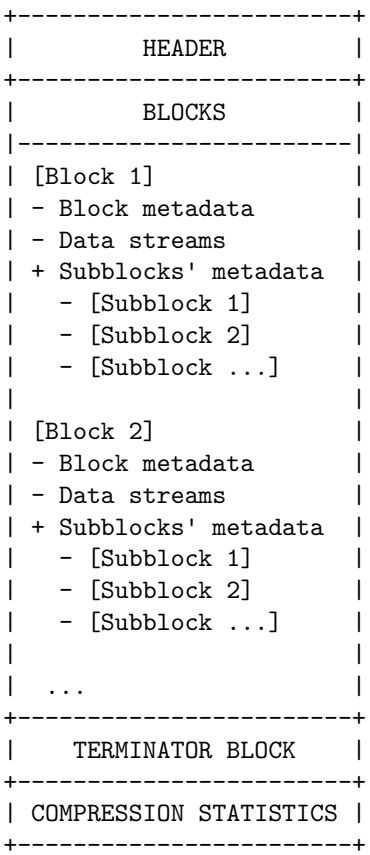
## 1.6 Subblock meta format

Within each block, the subblocks meta stream describes the logical subblocks (RAW, DNA, MIX, NNN) using 32-bit entries:

- The two most significant bits indicate the subblock type:
  - `00`: RAW (`RAW_SUBBLOCK_TYPE`)
  - `01`: DNA (`DNA_SUBBLOCK_TYPE`)
  - `10`: MIX (`MIX_SUBBLOCK_TYPE`)
  - `11`: NNN (`NNN_SUBBLOCK_TYPE`)
- The remaining 30 bits indicate the subblock size in bytes (i.e., number of decompressed symbols).

The subblock type (2 bits) can be extracted by applying the bitmask `0xC0000000` and right-shifting the result by 30 bits. The subblock size (30 bits) can be extracted by applying the bitmask `0x3FFFFFFF`.

## 1.7 File layout



## 1.8 Decompression

The decompressor first checks the `ffc_header` field (8 bytes) and proceeds upon correct verification.

Here we describe how one block is decompressed. As it consists of subblocks, the 32-bit int describing the subblock type and size is read, and further processing depends on the type:

- `DNA_SUBBLOCK_TYPE`, of size being a multiple of 8, then subblock_size / 4 bytes are consumed from the DNA stream and extracted into subblock_size DNA symbols,
- `MIX_SUBBLOCK_TYPE`, then subblock_size bytes are copied verbatim from the MIX stream to the output,
- `NNN_SUBBLOCK_TYPE`, then subblock_size 'N' characters are sent to the output,
- `RAW_SUBBLOCK_TYPE`, then subblock_size bytes are copied from the RAW stream to the output, similar as in case of `MIX_SUBBLOCK_TYPE`, with the extra End-Of-Line ('\n') character inserted (unless `block_meta.block_size` is already reached).

### 1.8.1 Handling EOLs

If `block_meta.seq_line_length` is 0, then there are no extra EOLs. Note that there can be some EOLs in RAW and MIX subblocks. Note also that after each RAW subblock one extra EOL is inserted, unless `block_meta.block_size` is already reached.

If `block_meta.seq_line_length` is positive, it determines locations of extra EOLs within subsequent DNA, MIX and NNN subblocks. The first EOL in the block is located at (`block_meta.first_EOL_offset`)-th byte, if `block_meta.first_EOL_offset` is from [0, `block_meta.block_size`) and its position is within a DNA, MIX or NNN subblock.

After the first EOL in the block or an extra EOL after each RAW subblock, the following extra EOLs are in regular distances to make each line length `block_meta.seq_line_length` bytes (not counting the terminating EOL itself). Note that there are no extra EOLs directly before RAW subblocks if the previous subblock is of type DNA, MIX or NNN.

### 1.8.2 Handling case flags

The case flag data of size `case_mask_compressed_size` bytes is applied to the whole block, with all the EOLs restored, of size approximately `case_mask_compressed_size` * 8 (approximately, since the last up to 63 bits of the case flags may be just padding). Each 8-byte chunk of case flag data corresponds to successive 64 bytes of block data, whose characters may be changed on their 5th bits (responsible for character case). The most significant bits of each byte in the case flag chunk are OR'ed with the 5th bits of the current 8-byte chunk of the block, the second most significant bits of each byte are OR'ed with the 5th bits of the next 8-byte chunk of the block, etc.

The pseudocode of the procedure:

```
result[0] = result[0] | ((flags & 0x0101010101010101) << 5);
result[1] = result[1] | ((flags & 0x0202020202020202) << 4);
result[2] = result[2] | ((flags & 0x0404040404040404) << 3);
result[3] = result[3] | ((flags & 0x0808080808080808) << 2);
result[4] = result[4] | ((flags & 0x1010101010101010) << 1);
result[5] = result[5] | ((flags & 0x2020202020202020)     );
result[6] = result[6] | ((flags & 0x4040404040404040) >> 1);
result[7] = result[7] | ((flags & 0x8080808080808080) >> 2);
```

where:

- flags - is 8-byte chunk of case flag data,
- result - is a corresponding output 8-element array of 8-byte chunks,
- `|`, `&`, `<<`, and `>>` - denote bitwise operators with C language semantics.