Nucleotide Archival Format (NAF) Specification

December 11, 2018

0 About this document

The latest version of this document can be found at https://github.com/KirillKryukov/naf . This document describes the Nucleotide Archival Format (NAF) version 1. This specification should allow re-implementing the NAF encoder and decoder. This specification is placed in the public domain.

Notation:

`Hexadecimal value`

1 Format overview

Nucleotide Archival Format (NAF) is a file format for storing DNA sequences. It's a binary format, storing zero or more sequences, with or without qualities. A NAF file typically contains sequences from a single FASTA or FASTQ file. NAF has no intrinsic limit on number of stored sequences, or on length of each sequence (although implementations may have their own limits). NAF supports IUPAC ambiguous nucleotide codes, as well as '-' for storing alignments. NAF also supports storing sequence mask, typically represented by lower case in FASTA files.

2 Overview

A NAF file may include some or all of these sections:

Header	Title	IDs	Names	Lengths	Mask	Sequence	Quality

Only "Header" is mandatory, all other sections are optional. The order of sections can't be changed.

3 Header

Header consists of the following fields:

Format descriptor	Format version	Flags	Name separator	Line length	Number of sequences

Header size is not fixed, because 'Line length' and 'Number of sequences' are stored in variable length encoding (described in section 10 of this specification).

2.1 Format descriptor

NAF format descriptor is a sequence of 3 bytes: `01 F9 EC` (in hexadecimal). A NAF file always begins with these bytes. This descriptor allows confirming the format of a NAF file, as

well as detecting the format of accidentally renamed NAF file (or anonymous chunk of data). This particular sequence of bytes is a reference to the "DNA Double Helix" symbol (🧬), recently added to Unicode as U+1F9EC.

2.2 Format version

A single byte stores the version of NAF format: `01`. This is a binary 1, not the text character '1'. The current document describes only this single version.

2.3 Flags

Flags determine what sections are present in a NAF file. It's a single byte, where each bit, when set, signifies the presence of the corresponding section. The meaning of each bit:

Bit	Meaning
`80`	Extended format
`40`	Title is present
`20`	IDs are present
`10`	Names are present
`08`	Lengths are present
`04`	Mask is present
`02`	Sequence is present
`01`	Quality is present

Bit 80 is reserved for future extension of the format. Currently it is always set to 0. A future backward-compatible extension of NAF will set this bit to 1.

2.4 Name separator

Name separator is a single byte. It stores the character that separates each sequence ID (accession number) from the rest of the name. A typical value is either `20` (' ', space) or `7C` ('|', pipe). These two characters are the ones most often used in databases. When decoding sequence names from a NAF file (e.g., when reconstructing a FASTA or FASTQ file), this character is inserted between sequence ID and the rest of sequence name, to produce the complete sequence name as it was in the original uncompressed file. Only printable ASCII characters are allowed as separators (characters with codes from `20` to `7E`, inclusive). (e.g., storing end-of-line character as separator would ensure broken decompressed output).

2.5 Line length

Line length is stored in variable length number encoding. When decoding NAF file into FASTA format, sequences are split into lines of this length. When encoding FASTA format into NAF, the reference encoder pragmatically stores the length of the longest sequence line. Storing only one line length for entire dataset means that only "well-formed" FASTA files can be restored

exact from NAF file. If the original FASTA file has inconsistent line lengths, such FASTA file can't be restored byte-to-byte identical to the original file.

2.6 Number of sequences

This is the number of individual sequences stored in the file. 0 is a valid number. It's stored using variable length number encoding. Although the maximum number of sequences is not restricted by the format, implementations may have their own limit.

3 Title

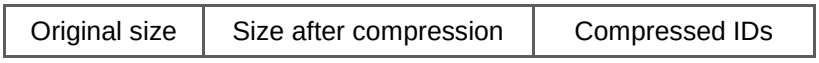
This is the title of the entire dataset. It's optional, and can be omitted (provided that the corresponding bit is cleared in header flags). Title format:

Size is in variable length number encoding. Text is a chunk of text, of exactly "Size" bytes. It should include only printable ASCII characters. It is not null-terminated.

4 IDs

IDs are normally some unique identifiers of sequences or reads in the dataset. (The unuiqueness is not required in this specification, but is a good idea anyway). In FASTA and FASTQ datasets these IDs are typically at the beginning of each complete sequence name. The IDs are stored separately from the rest of the names because: 1) It allows faster decompression when only IDs are required, and 2) It allows for slightly better overall compression.

The reference NAF encoder takes the part of sequence name before separator character as the ID. If the separator character is never found in the name, the entire complete sequence name then becomes the ID, which is totally ok. The '>' (for FASTA) or '@' (for FASTQ) character is not included as the part of the ID. The IDs are 0-terminated, concatenated together, and stored in the following format:



'Original size' and 'Size after compression' are stored in variable length number encoding. 'Compressed IDs' is the chunk of concatenated IDs (including null-termination of each ID), compressed using zstd and stored without the first 4 bytes. The first four bytes are omitted because they are always the same fixed "magic number". Each ID should contain only printable ASCII characters, and the number of IDs must match the number of sequences stored in the header. It's OK to have an empty ID, or even if all IDs are empty. In such case the IDs text before compression is simply a string of `00` bytes.

5 Names

The remaining part of the original sequence names, after exclusion of IDs, is stored in the "Names" section:

Original size	Size after compression	Compressed Names

The format is identical to that of the IDs section. During decompression, complete name of each sequence is reconstructed by concatenating: ID + separator character + Name. However in the case if the Name is empty, then the complete reconstructed name consists of just ID, without separator character.

6 Lengths

In the NAF format all sequences are stored concatenated together. Therefore sequence lengths are recorded separately in order to be able to reconstruct the original sequences. The lengths are stored in the format:

Original size	Size after compression	Compressed length units

'Original size' and 'Size after compression' are in variable length number encoding. 'Compressed length units' is the array of length units, compressed with zstd, and stored without the first 4 bytes. Length units encode sequence lengths in the following way: Each unit is a 32-bit unsigned integer value. Each length is represented by one of more units. If a length is smaller than 2^{32}-1 (4,294,967,295), then it is stored in just one unit. Otherwise, a unit of **`FF FF FF FF`** is written, and then the remaining part (original length minus (2^{32}-1)) is stored in subsequent one or more units. So each length is a sum of one or more consecutive units. This allows storing arbitrarily large lengths.

The NAF format does not restrict maximum sequence length, however encoder/decoder implementations and hardware may have their own restrictions. It's OK to have lengths of 0, even all of them. The total number of lengths (after decoding them from length units) must match the number of sequences stored in the header.

7 Mask

In FASTA files some part of sequence can be "masked", and stored in lower case. The normal, unmasked, sequence is typically in upper case. This mask usually represents repeats or low complexity sequence, but may have any other meaning. In NAF format the mask is stored separately from sequence. This allows making masking optional during decompression. Both decompressing and downstream processing is faster with unmasked sequence. The mask is stored in the format:

Original size	Size after compression	Compressed mask units

'Original size' and 'Size after compression' are in variable length number encoding. 'Compressed mask units' is the array of mask units, compressed with zstd, and stored without the first 4 bytes. Mask units encode mask intervals. A mask interval is the length of sequence with the same masking status (all masked or all unmasked). At the end of each mask interval, mask status

changes into opposite. The initial status is "unmasked" (capital letters). If the actual first sequence starts masked, then the first mask interval has to be 0.

Each mask unit is an 8-bit (1 byte) unsigned integer value. Each mask interval is represented by one of more units. If a mask interval is smaller than 255, then it is stored in single unit. Otherwise, a unit **FF** is written, and then the remaining part (original mask interval minus 255) is stored in subsequent one or more units. Each interval is a sum of one or more consecutive units, same as in case of sequence lengths. The total mask length (sum of all mask units) must be identical to the total sequence length (sum of all length units) (assuming both mask and lengths are stored in a NAF file).

8 Sequence

In NAF format all sequences are stored concatenated together into one long sequence. This sequence is then encoded in 4-bit encoding:

Nucleotide code	4-bit encoding
A	8
C	4
G	2
T	1
U	1
R	A
Y	5
S	6
W	9
K	3
M	C
B	7
D	B
H	D
V	E
N	F
-	0

This encoding allows storing ambiguous IUPAC nucleotide codes, including 'N'. Each of the 4 bits corresponds to a possible nucleotide:

Bit	Meaning
8	Can be 'A'

4	Can be 'C'
2	Can be 'G'
1	Can be 'T'

Different combinations of bits naturally represent different ambiguous or unambiguous nucleotide codes. 0 encodes gap ('-'), which allows storing alignments in NAF format.

Note that NAF format makes no distinction between 'T' and 'U'. If the original sequences file contained mixture of DNA and RNA (which should not happen in a well-formed FASTA file), the distinction will be lost after converting into NAF. Also, NAF format does not store whether the original file used 'T' or 'U'. The knowledge about whether the dataset contains DNA or RNA sequences is not stored in the NAF file, and has to be managed externally.

The 4-bit-encoded sequence is then stored in bytes in little-endian format. Each pair of 4-bit codes becomes one byte. The first code is stored in the lower half-byte (less significant), and the second code is stored in the higher half-byte (more significant). If the total sequence length is odd (not divisible by 2), then the last byte contains 0 in the higher half-byte.

This 4-bit-encoded sequence is then compressed with zstd and stored without the first 4 bytes, in format:

Sequence length	Size after compression	Compressed encoded sequence

'Sequence length' and 'Size after compression' are in variable length number encoding.

The reference encoder ignores any space and tab characters in the sequence in original FASTA/FASTQ files. Also it replaces any unrecognized characters (any characters other than the IUPAC codes in the table above) with N. The total sequence length must agree with the one recorded in the header.

9 Quality

All quality strings are concatenated and stored in the format:

Total length	Size after compression	Compressed quality

'Total length' and 'Size after compression' are in variable length number encoding. 'Compressed quality' is the entire quality text, compressed with zstd, and stored without the first 4 bytes. Total length must match the total length of sequences, and the length stored in the header.

10 Variable length number encoding

This encoding stores an unsigned integer number in variable space, depending on how large the number is. The encoding itself imposes no limit on how large number can be stored, although implementations may have their own limits. The number is converted into binary, then divided into "limbs" of 7 bits each (in the direction from least significant to most significant bits). Each

limb is stored in its own byte, in the order from most significant to least significant. All bytes except the last one have their highest bit (`80`) set to 1. This is analogous to storing the number in base 128 rather than base 10. Examples:

Number	Encoding
0	`00`
1	`01`
9	`09`
10	`0A`
100	`64`
127	`7F`
128	`81 00`
129	`81 01`
34,359,738,367	`FF FF FF FF 7F`
34,359,738,368	`81 80 80 80 80 00`

The rationale for using this encoding instead of fixed size numbers: 1. It occupies only 1 byte for small numbers, instead of 4 or 8 bytes typically used for fixed size numbers. 2. It has no upper limit for stored numbers, unlike the fixed size format.