

Sequence Alignment/Map Format Specification

The SAM/BAM Format Specification Working Group

25 Aug 2026

The master version of this document can be found at <https://github.com/samtools/hts-specs>.
This printing is version 7932c65 from that repository, last modified on the date shown above.

1 The SAM Format Specification

SAM stands for Sequence Alignment/Map format. It is a TAB-delimited text format consisting of a header section, which is optional, and an alignment section. If present, the header must be prior to the alignments. Header lines start with '@', while alignment lines do not. Each alignment line has 11 mandatory fields for essential alignment information such as mapping position, and variable number of optional fields for flexible or aligner specific information.

This specification is for version 1.6 of the SAM and BAM formats. Each SAM and BAM file may optionally specify the version being used via the @HD VN tag. For full version history see Appendix B.

SAM file contents are 7-bit US-ASCII, except for certain field values as individually specified which may contain other Unicode characters encoded in UTF-8. Alternatively and equivalently, SAM files are encoded in UTF-8 but non-ASCII characters are permitted only within certain field values as explicitly specified in the descriptions of those fields.¹

Where it makes a difference, SAM file contents should be read and written using the POSIX / C locale. For example, floating-point values in SAM always use '.' for the decimal-point character.

The regular expressions in this specification are written using the POSIX / IEEE Std 1003.1 extended syntax.

1.1 An example

Suppose we have the following alignment with bases in lowercase clipped from the alignment. Read r001/1 and r001/2 constitute a read pair; r003 is a chimeric read; r004 represents a split alignment.

```
Coor      12345678901234 5678901234567890123456789012345
ref       AGCATGTTAGATAA**GATAGCTGTGCTAGTAGGCAGTCAGCGCCAT

+r001/1      TTAGATAAAGGATA*CTG
+r002        aaaAGATAA*GGATA
+r003        gcctaAGCTAA
+r004                ATAGCT.....TCAGC
-r003                ttagctTAGGC
-r001/2                        CAGCGGCAT
```

¹Hence in particular SAM files must not begin with a byte order mark (BOM) and lines of text are delimited by ASCII line terminator characters only. In addition to the local platform's text file line termination conventions, implementations may wish to support LF and ~~CR-LF~~ CR-LF for interoperability with other platforms.

DT	Date the run was produced (ISO8601 date or date/time).
FO	Flow order. The array of nucleotide bases that correspond to the nucleotides used for each flow of each read. Multi-base flows are encoded in IUPAC format, and non-nucleotide flows by various other characters. <i>Format:</i> /* [ACMGRSVTWYHKDBN] +/
KS	The array of nucleotide bases that correspond to the key sequence of each read.
LB	Library.
PG	Programs used for processing the read group.
PI	Predicted median insert size, rounded to the nearest integer.
PL	Platform/technology used to produce the reads. <i>Valid values:</i> CAPILLARY, DNBSEQ (MGI/BGI), ELEMENT, HELICOS, ILLUMINA, IONTORRENT, LS454, ONT (Oxford Nanopore), PACBIO (Pacific Biosciences), SINGULAR, SOLID, and ULTIMA. ¹¹ This field should be omitted when the technology is not in this list (though the PM field may still be present in this case) or is unknown.
PM	Platform model. Free-form text providing further details of the platform/technology used.
PU	Platform unit (e.g., flowcell-barcode.lane for Illumina or slide for SOLiD). Unique identifier.
SM	Sample. Use pool name where a pool is being sequenced.
@PG	Program.
ID*	Program record identifier. Each @PG line must have a unique ID. The value of ID is used in the alignment PG tag and PP tags of other @PG lines. PG IDs may be modified when merging SAM files in order to handle collisions.
PN	Program name
CL	Command line. UTF-8 encoding may be used.
PP	Previous @PG-ID. Must match another @PG header's ID tag. @PG records may be chained using PP tag, with the last record in the chain having no PP tag. This chain defines the order of programs that have been applied to the alignment. PP values may be modified when merging SAM files in order to handle collisions of PG IDs. The first PG record in a chain (i.e., the one referred to by the PG tag in a SAM record) describes the most recent program that operated on the SAM record. The next PG record in the chain describes the next most recent program that operated on the SAM record. The PG ID on a SAM record is not required to refer to the newest PG record in a chain. It may refer to any PG record in a chain, implying that the SAM record has been operated on by the program in that PG record, and the program(s) referred to via the PP tag.
DS	Description. UTF-8 encoding may be used.
VN	Program version
@CO	One-line text comment. Unordered multiple @CO lines are allowed. UTF-8 encoding may be used.

1.3.1 Defined sub-sort terms

While the SS sub-sort field allows implementation-defined keywords, some terms are predefined with specific meanings.

lexicographical sort order is defined as a character-based dictionary sort with the character order as defined by the POSIX C locale. For example “abc”, “abc17”, “abc5”, “abc59” and “abcd” are in lexicographical order.

natural sort order is similar to lexicographical order except that runs of adjacent digits are considered to be numbers embedded within the text string, ordered numerically when compared to each other and ordered as single digits when compared to the surrounding non-digit characters. Runs that differ only in the number of leading zeros (thus are numerically tied) are ordered by more-zeros coming before fewer-zeros. The characters ‘-’ and ‘.’ are considered as ordinary characters, so apparently negative or fractional values are not treated as part of an embedded number. For example, “abc”, “abc+5”, “abc-5”, “abc.d”, “abc03”, “abc5”, “abc008”, “abc08”, “abc8”, “abc17”, “abc17.+”, “abc17.2”, “abc17.d”, “abc59” and “abcd” are in natural order.

¹¹[The PL value should be written in uppercase exactly as shown in this list of valid values. Tools should also accept lowercase when reading the @RG PL field, due to the existence of public data files with lowercase PL values.](#)

but the two ends must still have differing signs. It is set as 0 for a single-segment template or when the information is unavailable (e.g., when the first or last segment of a multi-segment template is unmapped or when the two are mapped to different reference sequences).

The intention of this field is to indicate where the other end of the template has been aligned without needing to read the remainder of the SAM file. Unfortunately there has been no clear consensus on the definitions of the template mapped start and end. Thus the exact definitions are implementation-defined.¹⁶

10. **SEQ**: segment SEQUENCE. This field can be a '*' when the sequence is not stored. If not a '*', the length of the sequence must equal the sum of lengths of M/I/S/=X operations in CIGAR. An '=' denotes the base is identical to the reference base. No assumptions can be made on the letter cases.
11. **QUAL**: ASCII of base QUALity plus 33 (same as the quality string in the Sanger FASTQ format). A base quality is the phred-scaled base error probability which equals $-10 \log_{10} \Pr\{\text{base is wrong}\}$. This field can be a '*' when quality is not stored.¹⁷ If not a '*', SEQ must not be a '*' and the length of the quality string ought to equal the length of SEQ.

1.5 The alignment section: optional fields

All optional fields follow the TAG:TYPE:VALUE format where TAG is a two-character string that matches `/[A-Za-z][A-Za-z0-9]/`. Within each alignment line, no TAG may appear more than once and the order in which the optional fields appear is not significant. A TAG containing lowercase letters is reserved for end users. In an optional field, TYPE is a single case-sensitive letter which defines the format of VALUE:

Type	Regexp matching VALUE	Description
A	<code>[!~]</code>	Printable character
i	<code>[+]?[0-9]+</code>	Signed integer ¹⁸
f	<code>[+]?[0-9]*\.[0-9]+([eE][+]?[0-9]+)?</code>	Single-precision floating number
Z	<code>[!~]*</code>	Printable string, including space
H	<code>([0-9A-F][0-9A-F])*</code>	Byte array in the Hex format ¹⁹
B	<code>[cCsSiIf](,[+]?[0-9]*\.[0-9]+([eE][+]?[0-9]+)?)*</code>	Integer or numeric array

For an integer or numeric array (type 'B'), the first letter indicates the type of numbers in the following comma separated array. The letter can be one of 'cCsSiIf', corresponding to `int8_t` (signed 8-bit integer), `uint8_t` (unsigned 8-bit integer), `int16_t`, `uint16_t`, `int32_t`, `uint32_t` and `float`, respectively.²⁰ During import/export, the element type may be changed if the new type is also compatible with the array.

Predefined tags are described in the separate *Sequence Alignment/Map Optional Fields Specification*.²¹ See that document for details of existing standard tag fields and conventions around creating new tags that may be of general interest. Tags starting with 'X', 'Y' or 'Z' and tags containing lowercase letters in either position are reserved for local use and will not be formally defined in any future version of these specifications.

¹⁶The earliest versions of this specification used 5' to 5' (in original orientation, TLEN#1; dashed parts of the reads indicate soft-clipped bases) while later ones used leftmost to rightmost mapped base (TLEN#2). Note: these two definitions agree in most alignments, but differ in the case of overlaps where the first segment aligns beyond the start of the last segment.



¹⁷In the unlikely case in which SEQ is of length 1 and QUAL is '*', QUAL should be interpreted as quality not stored. Tools should avoid writing a solitary base quality 9 (ASCII '*') to SAM, BAM, or CRAM files for alignment lines where SEQ is of length 1, instead adjusting the base quality to 10 (ASCII '+') to avoid this potential ambiguity.

¹⁸The number of digits in an integer optional field is not explicitly limited in SAM. However, BAM can represent values in the range $[-2^{31}, 2^{32}]$, so in practice this is the realistic range of values for SAM's 'i' as well.

¹⁹For example, the six-character Hex string '1AE301' represents the byte array `[0x1a, 0xe3, 0x01]`.

²⁰Explicit typing eases format parsing and helps to reduce the file size when SAM is converted to BAM.

²¹See `SAMtags.pdf` at <https://github.com/samtools/hts-specs>.

2 Recommended Practice for the SAM Format

This section describes the best practice for representing data in the SAM format. They are not required in general, but may be required by a specific software package for it to function properly.

1. The header section

- 1 The `@HD` line should be present, with either the `SO` tag or the `GO` tag (but not both) specified.
- 2 The `@SQ` lines should be present if reads have been mapped.
- 3 When a `RG` tag appears anywhere in the alignment section, there should be a single corresponding `@RG` line with matching `ID` tag in the header.
- 4 When a `PG` tag appears anywhere in the alignment section, there should be a single corresponding `@PG` line with matching `ID` tag in the header.

2. Adjacent CIGAR operations should be different.

3. No alignments should be assigned mapping quality 255.

4. Unmapped reads

- 1 For a unmapped paired-end or mate-pair read whose mate is mapped, the unmapped read should have `RNAME` and `POS` identical to its mate.
- 2 If all segments in a template are unmapped, their `RNAME` should be set as `*` and `POS` as 0.
- 3 If `POS` plus the sum of lengths of `M/=X/D/N` operations in `CIGAR` exceeds the length specified in the `LN` field of the `@SQ` header line (if exists) with an `SN` equal to `RNAME`, the alignment should be unmapped, unless the reference sequence is circular (see below).
- 4 Unmapped reads should be stored in the orientation in which they came off the sequencing machine and have their `reverse` flag bit (0x10) correspondingly unset.
- 5 Fields stated to have no assumptions on their content if FLAG bit 0x4 is set (unmapped) should have values `*` or `0` as appropriate. ²²
Specifically this includes `RNAME` (`*`, but see caveat), `POS` (`0`, see caveat), `CIGAR` (`*`), `MAPQ` (`0`) and FLAG bits 0x2, 0x100 and 0x800 (all unset).
One caveat to this is with paired sequencing protocols where one record is aligned and one unaligned within a pair. See item 1 above.

5. Multiple mapping

- 1 When one segment is present in multiple lines to represent a multiple mapping of the segment, only one of these records should have the secondary alignment flag bit (0x100) unset. `RNEXT` and `PNEXT` point to the primary line of the next read in the template.
- 2 `SEQ` and `QUAL` of secondary alignments should be set to `*` to reduce the file size.

6. Optional tags:

- 1 If the template has more than 2 segments, the `TC` tag should be present.
- 2 The `NM` tag should be present.

7. Circular reference sequences

Mappings that cross the coordinate 'join' in circular reference sequences (i.e., those whose `@SQ` headers specify `TP:circular`) may be represented as follows:

²²This is desirable as CRAM will not store some or all of these fields for unmapped data, so round-trips should yield these default values for maximum compatibility.

4 The BAM Format Specification

4.1 The BGZF compression format

BGZF is block compression implemented on top of the standard gzip file format.²⁷ The goal of BGZF is to provide good compression while allowing efficient random access to the BAM file for indexed queries. The BGZF format is ‘gzip compatible’, in the sense that a compliant gzip utility can decompress a BGZF compressed file.²⁸

A BGZF file is a series of concatenated BGZF blocks, each no larger than ~~64Kb before or~~ 64 KiB both before and after compression. Each BGZF block is itself a spec-compliant gzip archive which contains an “extra field” in the format described in RFC1952. The gzip file format allows the inclusion of application-specific extra fields and these are ignored by compliant decompression implementation. The gzip specification also allows gzip files to be concatenated. The result of decompressing concatenated gzip files is the concatenation of the uncompressed data.

Each BGZF block contains a standard gzip file header with the following standard-compliant extensions:

1. The F.EXTRA bit in the header is set to indicate that extra fields are present.
2. The extra field used by BGZF uses the two subfield ID values 66 and 67 (ASCII ‘BC’).
3. The length of the BGZF extra field payload (field LEN in the gzip specification) is 2 (two bytes of payload).
4. The payload of the BGZF extra field is a 16-bit unsigned integer in little endian format. This integer gives the size of the containing BGZF block minus one.

On disk, a complete BGZF file is a series of blocks as shown in the following table. (All integers are little endian as is required by RFC1952.)

Field	Description	Type	Value
<i>List of compression blocks (until the end of the file)</i>			
ID1	gzip IDentifier1	uint8_t	31
ID2	gzip IDentifier2	uint8_t	139
CM	gzip Compression Method	uint8_t	8
FLG	gzip FLaGs	uint8_t	4
MTIME	gzip Modification TIME	uint32_t	
XFL	gzip eXtra FLags	uint8_t	
OS	gzip Operating System	uint8_t	
XLEN	gzip eXtra LENgth	uint16_t	
<i>Extra subfield(s) (total size=XLEN)</i>			
<i>Additional RFC1952 extra subfields if present</i>			
SI1	Subfield Identifier1	uint8_t	66
SI2	Subfield Identifier2	uint8_t	67
SLEN	Subfield LENgth	uint16_t	2
BSIZE	total Block SIZE minus 1	uint16_t	
<i>Additional RFC1952 extra subfields if present</i>			
CDATA	Compressed DATA by zlib::deflate()	uint8_t[BSIZE-XLEN-19]	
CRC32	CRC-32	uint32_t	
ISIZE	Input SIZE (length of uncompressed data)	uint32_t	

The random access method to be described next limits the uncompressed contents of each BGZF block to a maximum of 2^{16} bytes of data. Thus while ISIZE is stored as a uint32_t as per the gzip format, in BGZF it is limited to the range [0,65536]. BSIZE can represent BGZF block sizes in the range [1,65536], though typically BSIZE will be rather less than ISIZE due to compression.

²⁷L. Peter Deutsch, **GZIP file format specification version 4.3**, RFC 1952.

²⁸It is worth noting that there is a known bug in the Java GZIPInputStream class that concatenated gzip archives cannot be successfully decompressed by this class. BGZF files can be created and manipulated using the built-in Java util.zip package, but naive use of GZIPInputStream on a BGZF file will not work due to this bug.