Sequence Alignment/Map Format Specification

The SAM/BAM Format Specification Working Group

29 Jan 2024

The master version of this document can be found at `https://github.com/samtools/hts-specs`.
This printing is version a8550ef 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
```

[1] Hence 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.

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.[15]

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. If not a '*', SEQ must not be a '*' and the length of the quality string ought to equal the length of SEQ. [16]

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`:

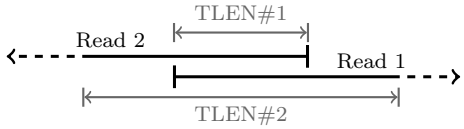
Type	Regexp matching `VALUE`	Description
A	`[!-~]`	Printable character
i	`[-+]?[0-9]+`	Signed integer[17]
f	`[-+]?[0-9]*\.?[0-9]+([eE][-+]?[0-9]+)?`	Single-precision floating number
Z	`[ !-~]*`	Printable string, including space
H	`([0-9A-F][0-9A-F])*`	Byte array in the Hex format[18]
B	`[cCsSiIf](,[-+]?[0-9]*\.?[0-9]+([eE][-+]?[0-9]+)?)*`	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.[19] 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*.[20] 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.

[15] The 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.

Unambiguous scenario



Ambiguous scenario

[16] There is a small ambiguity with a sequence exactly one base-pair long having quality 9. This is ASCII '*' so it could be interpreted as either QUAL 9 or as quality unavailable. Given this ambiguity, we recommend interpreting it as unavailable.

[17] The 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.

[18] For example, the six-character Hex string '1AE301' represents the byte array [0x1a, 0xe3, 0x1].

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

[20] See `SAMtags.pdf` at `https://github.com/samtools/hts-specs`.