Sequence Alignment/Map Format Specification

The SAM/BAM Format Specification Working Group

13 Sep 2022

The master version of this document can be found at `https://github.com/samtools/hts-specs`.
This printing is version 3d3da3a 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.

~~Unless explicitly specified elsewhere, all fields are encoded~~ SAM files are encoded in UTF-8. They must not begin with a byte order mark, and non-ASCII characters are permitted only in certain field values as individually specified.[1] SAM file contents should be read and written using ~~7-bit US-ASCII~~ [2] ~~in using~~ the POSIX / C locale.~~Regular expressions listed use~~ [2] The regular expressions in this specification have been 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  56789012345678901234567890123456
ref      AGCATGTTAGATAA**GATAGCTGTGCTAGTAGGCAGTCAGCGCCAT

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

The corresponding SAM format is:[3]

[1] Equivalently, SAM files primarily contain US-ASCII characters in the usual single-byte encoding; certain field values as specified may contain other Unicode characters and are encoded as UTF-8.

[2] ~~Charset ANSI_X3.4-1968 as defined in RFC1345.~~

[2] For example, floating-point values in SAM always use '.' (FULL STOP) for the decimal-point character.

[3] The values in the FLAG column correspond to bitwise flags as follows: 99 = 0x63: first/next is reverse-complemented/ properly aligned/multiple segments; 0: no flags set, thus a mapped single segment; 2064 = 0x810: supplementary/reverse-complemented; 147 = 0x93: last (second of a pair)/reverse-complemented/properly aligned/multiple segments.

	PL	Platform/technology used to produce the reads. *Valid values*: CAPILLARY, DNBSEQ (MGI/BGI), ELEMENT, HELICOS, ILLUMINA, IONTORRENT, LS454, ONT (Oxford Nanopore), PACBIO (Pacific Biosciences), ~~and~~ SOLID, and ULTIMA. 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.

umi is a lexicographical sort by the UMI tag. The MI tag should be used for comparing UMIs. The RX tag may be used in its absence but is not guaranteed to be unique across multiple libraries.

1.3.2 Reference MD5 calculation

The M5 tag on @SQ lines allows reference sequences to be uniquely identified through the MD5 digest of the sequence itself. As the digest is based on the sequence and nothing else, it can help resolve ambiguities with reference naming. For example, it allows a quick way of checking that references named '1', 'Chr1' and 'chr1' in different files are in fact the same.

Appendix A Parsing region notation

Parsing region notation such as *name*`[:`*begin*`[-`*end*`]]` (in which omission of the outer bracketed portion indicates a request for the entire reference sequence) would be simple if *name* could not itself contain '`:`' characters, but this is not the case. (No such notation containing an optional '`:`' appears in the SAM format itself, but various tools use this notation as a convenient way for their users to specify regions of interest.)

The set of valid reference sequence names is usually already known when parsing this notation—for example, because the associated `@SQ` headers have already been encountered. Tools can use this set to determine unambiguously which colons could delimit a known-valid reference sequence name.

In pseudocode form, a string *str* can be parsed as follows:

consider the rightmost '`:`' character, if any, of *str*
if *str* is of the form '*prefix*`:NUM`' or '*prefix*`:NUM-NUM`'
or generally '*prefix*`:`*suffix*' for some plausible interval suffix
then
if both *prefix* and *str* are in the known set then ... error: ambiguous representation
else if *prefix* is in the known set then return (*prefix*, `NUM...NUM`)
else if *str* is in the known set then return (*str*, entire sequence)
else ... error: unknown reference sequence name

else ... either *str* does not contain a colon or the suffix is not plausibly numeric
if *str* is in the known set then return (*str*, entire sequence)
else ... error: unknown reference sequence name or invalid interval syntax

The check leading to "error: ambiguous representation" is important as it prevents confusing interpretations of actually ambiguous input. Typically the set of valid reference sequence names will not contain names that are prefixes of other names in the set, so in practice this error will not usually be encountered in non-malicious data.

Either in addition to this algorithm or as an alternative to it, tools can use additional delimiter characters to make an unambiguously parsable notation. We recommend a convention using curly brackets around the reference sequence name— `{`*name*`}[:`*begin*`[-`*end*`]]` —as being memorable, easily typed, unambiguous, and not expanded by most shells.

Appendix B SAM Version History

This lists the date of each tagged SAM version along with changes that have been made while that version was current. The key changes that caused the version number to change are shown in bold.

Additions and changes to the standard predefined tags are listed in the separate *Sequence Alignment/Map Optional Fields Specification*.[38]

1.6: 28 November 2017 to current

- Add `ELEMENT` and `ULTIMA` to the list of `@RG PL` header tag values. (Aug 2022)
- Clarify that header field tags must be distinct within each line, and that the ordering of both header fields and alignment optional fields is not significant. (Jun 2021)
- Clarify the meaning of TLEN when secondary alignments are present. (May 2021)
- Bin calculation changed for alignment records whose CIGAR strings consume no reference bases: like unmapped records, they are considered to have length one (rather than zero). (Jan 2021)
- Correct the description of index pseudo-bins, which previously stated that ref_beg/ref_end, then named unmapped_beg/unmapped_end, include only placed unmapped reads. (Jul 2020)

[38] See Appendix A of `SAMtags.pdf` at `https://github.com/samtools/hts-specs`.