Sequence Alignment/Map Format Specification

The SAM/BAM Format Specification Working Group

17 Jun 2025

The master version of this document can be found at `https://github.com/samtools/hts-specs`.
This printing is version 35bc658 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 using~~ SAM file contents are 7-bit US-ASCII[1] ~~in~~ , 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. ~~Regular expressions listed use the POSIX~~ 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] ~~Charset ANSI_X3.4-1968 as defined in RFC1345.~~

[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 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. *Format*: `/\*|[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. *Valid values*: `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.

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.

The reference sequence must be in the 7-bit US-ASCII character set. All valid reference bases can be represented in this set, and it avoids the problem of determining exactly which 8-bit representation may have been used. Padding characters (See Section 3.2) must be represented only using the '*' character.

The digest is calculated as follows:

- All characters outside of the inclusive range 33 ('!') to 126 ('~') are stripped out. This removes all unprintable and whitespace characters including spaces and new lines. Everything else is retained, even if not a legal nucleotide code.
- All lowercase characters are converted to uppercase. This operation is equivalent to calling *toupper()* on characters in the *POSIX locale*.
- The MD5 digest is calculated as described in *RFC 1321* and presented as a 32 character lowercase hexadecimal number.

As an example, if the reference contains the following characters (including spaces):

```
ACGT ACGT ACGT
acgt acgt acgt
... 12345 !!!
```

then the digest is that of the string `ACGTACGTACGTACGTACGTACGT...12345!!!` and the resulting tag would be `M5:dfabdbb36e239a6da88957841f32b8e4`.

In padded SAM files, the padding bases should be inserted into the reference as '*' characters. Taking the example in Section 3.2, the padded version of the reference is

```
AGCATGTTAGATAA**GATAGCTGTGCTAGTAGGCAGTCAGCGCCAT
```

and the corresponding tag is `M5:caad65b937c4bc0b33c08f62a9fb5411`.

1.4 The alignment section: mandatory fields

In the SAM format, each alignment line typically represents the linear alignment of a segment. Each line consists of 11 or more TAB-separated fields. The first eleven fields are always present and in the order shown below; if the information represented by any of these fields is unavailable, that field's value will be a placeholder, either '0' or '*' as determined by the field's type. The following table gives an overview of these mandatory fields in the SAM format:

Col	Field	Type	Regexp/Range	Brief description
1	QNAME	String	`[!-?A-~]{1,254}`	Query template NAME
2	FLAG	Int	$[0, 2^{16} - 1]$	bitwise FLAG
3	RNAME	String	`\*|[:rname:^*=][:rname:]*`	Reference sequence NAME[12]
4	POS	Int	$[0, 2^{31} - 1]$	1-based leftmost mapping POSition
5	MAPQ	Int	$[0, 2^{8} - 1]$	MAPping Quality
6	CIGAR	String	`\*|([0-9]+[MIDNSHPXMIDNSHP=X])+`	CIGAR string
7	RNEXT	String	`\*|=|[:rname:^*=][:rname:]*`	Reference name of the mate/next read
8	PNEXT	Int	$[0, 2^{31} - 1]$	Position of the mate/next read
9	TLEN	Int	$[-2^{31} + 1, 2^{31} - 1]$	observed Template LENgth
10	SEQ	String	`\*|[A-Za-z=.]+`	segment SEQuence
11	QUAL	String	`[!-~]+`	ASCII of Phred-scaled base QUALity+33

3. RNAME: Reference sequence NAME of the alignment. If `@SQ` header lines are present, RNAME (if not '*') must be present in one of the `SQ-SN` tag. An unmapped segment without coordinate has a '*' at this field. However, an unmapped segment may also have an ordinary coordinate such that it can be placed at a desired position after sorting. If RNAME is '*', no assumptions can be made about POS and CIGAR.

4. POS: 1-based leftmost mapping POSition of the first CIGAR operation that "consumes" a reference base (see table below). The first base in a reference sequence has coordinate 1. POS is set as 0 for an unmapped read without coordinate. If POS is 0, no assumptions can be made about RNAME and CIGAR.

5. MAPQ: MAPping Quality. It equals $-10\log_{10}\Pr\{\text{mapping position is wrong}\}$, rounded to the nearest integer. A value 255 indicates that the mapping quality is not available.

6. CIGAR: CIGAR string. The CIGAR operations are given in the following table (set '*' if unavailable):

Op	BAM	Description	Consumes query	Consumes reference
`M`	0	alignment match (can be a sequence match or mismatch)	yes	yes
`I`	1	insertion to the reference	yes	no
`D`	2	deletion from the reference	no	yes
`N`	3	skipped region from the reference	no	yes
`S`	4	soft clipping (clipped sequences present in SEQ)	yes	no
`H`	5	hard clipping (clipped sequences NOT present in SEQ)	no	no
`P`	6	padding (silent deletion from padded reference)	no	no
`=`	7	sequence match	yes	yes
`X`	8	sequence mismatch	yes	yes

   - "Consumes query" and "consumes reference" indicate whether the CIGAR operation causes the alignment to step along the query sequence and the reference sequence respectively.
   - `H` can only be present as the first and/or last operation.
   - `S` may only have `H` operations between them and the ends of the CIGAR string.
   - For mRNA-to-genome alignment, an `N` operation represents an intron. For other types of alignments, the interpretation of `N` is not defined.
   - Sum of lengths of the ~~M/I/S/=/X~~`M/I/S/=/X` operations shall equal the length of SEQ.

7. RNEXT: Reference sequence name of the primary alignment of the NEXT read in the template. For the last read, the next read is the first read in the template. If `@SQ` header lines are present, RNEXT (if not '*' or '=') must be present in one of the `SQ-SN` tag. This field is set as '*' when the information is unavailable, and set as '=' if RNEXT is identical RNAME. If not '=' and the next read in the template has one primary mapping (see also bit 0x100 in FLAG), this field is identical to RNAME at the primary line of the next read. If RNEXT is '*', no assumptions can be made on PNEXT and bit 0x20.

8. PNEXT: 1-based Position of the primary alignment of the NEXT read in the template. Set as 0 when the information is unavailable. This field equals POS at the primary line of the next read. If PNEXT is 0, no assumptions can be made on RNEXT and bit 0x20.

9. TLEN: signed observed Template LENgth. For primary reads where the primary alignments of all reads in the template are mapped to the same reference sequence, the absolute value of TLEN equals the distance between the mapped end of the template and the mapped start of the template, inclusively (i.e., end − start + 1).[15] Note that *mapped base* is defined to be one that aligns to the reference as described by CIGAR, hence excludes soft-clipped bases. The TLEN field is positive for the leftmost segment of the template, negative for the rightmost, and the sign for any middle segment is undefined. If segments cover the same coordinates then the choice of which is leftmost and rightmost is arbitrary,

[15] Thus a segment aligning in the forward direction at base 100 for length 50 and a segment aligning in the reverse direction at base 200 for length 50 indicate the template covers bases 100 to 249 and has length 150.

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~~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`:

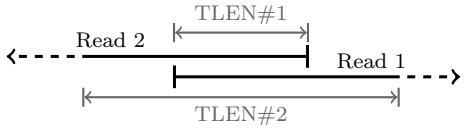
Type	Regexp matching `VALUE`	Description
A	`[!-~]`	Printable character
i	`[-+]?[0-9]+`	Signed integer[18]
f	`[-+]?[0-9]*\.?[0-9]+([eE][-+]?[0-9]+)?`	Single-precision floating number
Z	`[ !-~]*`	Printable string, including space
H	`([0-9A-F][0-9A-F])*`	Byte array in the Hex format[19]
B	`[cCsSiIf](,[-+]?[0-9]*\.?[0-9]+([eE][-+]?[0-9]+)?)*`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

[16] 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

[17] In 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.

[18] 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.

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

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

[21] See `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~~`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. 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:
    1. (Preferred) As usual POS should be between 1 and the `@SQ` header's `LN` value, but POS plus the sum of the lengths of `M/=/X/D/N` ~~M/=/X/D/N~~CIGAR operations may exceed `LN`. Coordinates greater than `LN` are interpreted by subtracting `LN` so that bases at $\texttt{LN}+1, \texttt{LN}+2, \texttt{LN}+3, \ldots$ are considered to be mapped at positions $1, 2, 3, \ldots$; thus each (1-based) position p is interpreted as $((p-1) \bmod \texttt{LN}) + 1$.[22]
    2. Alternatively, such alignments may be split across several records: one record representing the initial portion of the segment ending at `LN`, one representing the final portion starting from 1, and any other records representing additional portions in between spanning the entire reference sequence. One record (chosen arbitrarily) is considered primary and the remainder have their supplementary flag bit (0x800) set.

[22] The impact of this representation on indexing and random access is yet to be explored by implementations.

4.2 The BAM format

BAM is compressed in the BGZF format. All multi-byte numbers in BAM are little-endian, regardless of the machine endianness. The format is formally described in the following table where values in brackets are the default when the corresponding information is not available; an underlined word in uppercase denotes a field in the SAM format.

Field	Description	Type	Value
magic	BAM magic string	char[4]	BAM\1
l_text	Length of the header text, including any NUL padding	uint32_t	$< 2^{31}$
text	Plain header text in SAM; not necessarily NUL-terminated	char[l_text]	
n_ref	# reference sequences	uint32_t	$< 2^{31}$
List of reference information (n=n_ref)			
l_name	Length of the reference name plus 1 (including NUL)	uint32_t	*limited*
name	Reference sequence name; NUL-terminated	char[l_name]	
l_ref	Length of the reference sequence	uint32_t	$< 2^{31}$
List of alignments (until the end of the file)			
block_size	Total length of the alignment record, excluding this field	uint32_t	*limited*
refID	Reference sequence ID, $-1 \leq$ refID $<$ n_ref; -1 for a read without a mapping position	int32_t	[-1]
pos	0-based leftmost coordinate (= POS − 1)	int32_t	[-1]
l_read_name	Length of read_name below (= length(QNAME) + 1)	uint8_t	
mapq	Mapping quality (=MAPQ)	uint8_t	
bin	BAI index bin, see Section 4.2.1	uint16_t	
n_cigar_op	Number of operations in CIGAR, see Section 4.2.2	uint16_t	
flag	Bitwise flags (= FLAG)[31]	uint16_t	
l_seq	Length of SEQ	uint32_t	*limited*
next_refID	Ref-ID of the next segment ($-1 \leq$ next_refID $<$ n_ref)	int32_t	[-1]
next_pos	0-based leftmost pos of the next segment (= PNEXT − 1)	int32_t	[-1]
tlen	Template length (= TLEN)	int32_t	[0]
read_name	Read name, NUL-terminated (QNAME with trailing '\0')[32]	char[l_read_name]	
cigar	CIGAR: op_len<<4|op. 'MIDNSHP=X'→'012345678'	uint32_t[n_cigar_op]	
seq	4-bit encoded read: '=ACMGRSVTWYHKDBN'→ [0, 15]. See Section 4.2.3	uint8_t[(l_seq+1)/2]	
qual	Phred-scaled base qualities. See Section 4.2.3	char[l_seq]	
List of auxiliary data (until the end of the alignment block)			
tag	Two-character tag	char[2]	
val_type	Value type: AcCsSiIfZHB, see Section 4.2.4	char	
value	Tag value	(by val_type)	

Most length and count fields described as uint32_t have additional constraints on their range: l_text $< 2^{31}$ due to implementation limits; n_ref $< 2^{31}$ because refID and next_refID are signed; l_ref $< 2^{31}$ because tlen is signed; those marked "*limited*" are limited by available memory and the practical size of the data represented well before they are limited by, e.g., Java's signed 32-bit integer maximum array size.

4.2.1 BIN field calculation

BIN is calculated using the reg2bin() function in Section 5.3. For mapped reads this uses POS−1 (i.e., 0-based left position) and the alignment end point using the alignment length from the CIGAR string. For unmapped reads (e.g., paired-end reads where only one part is mapped, see Section 2) and reads whose CIGAR strings consume no reference bases at all, the alignment is treated as being of length one. Note unmapped reads with POS 0 (which becomes −1 in BAM) therefore use reg2bin(−1, 0) which is computed as 4680.

[31] As noted in Section 1.4, reserved FLAG bits should be written as zero and ignored on reading by current software.

[32] For backward compatibility, an absent QNAME (represented as '*' in SAM) is stored as a C string "*\0".

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*.[39]

1.6: 28 November 2017 to current

- Add `SINGULAR` to the list of `@RG PL` header tag values. (May 2023)
- Clarify that `@RG PI` values are integers. (May 2023)
- 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)

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

1.4: 21 April 2011 to May 2013

- Add guide to using sequence annotations (`CT/PT` tags). (Mar 2012)
- Increase max reference length from 2^{29} to 2^{31}. (Sep 2011)
- Clarify `@SQ M5` header tag generation. (Sep 2011)
- Describe padded alignments. (Sep 2011)
- Add `@RG FO`, `KS` header fields. (Apr 2011)
- Clarify chaining of PG records. (Apr 2011)
- **Add `B` array auxiliary tag type.** (Apr 2011)
- **Permit IUPAC in SEQ and `MD` auxiliary tag.** (Apr 2011)
- **Permit QNAME "*".** (Apr 2011)

1.3: July 2010 to April 2011

- Add `RG PG` header field. (Nov 2010)
- Add BAM description and index sections. (Nov 2010)
- **Add '=' and 'X' CIGAR operations.** (July 2010)
- **Removal of FLAG letters.** (July 2010)
- The `SM` header field, previously mandatory for `@RG`, is now optional. (July 2010)

1.0: 2009 to July 2010

Initial edition.