Key	Value data type	Name	Value
BF	encoding<int>	BAM bit flags	see separate section
CF	encoding<int>	CRAM bit flags	see specific section
RI	encoding<int>	reference id	record reference id from the SAM file header
RL	encoding<int>	read lengths	read lengths
AP	encoding<int>	in-seq positions	if **AP-Delta** = true: 0-based alignment start delta from the AP value in the previous record. Note this delta may be negative, for example when switching references in a multi-reference slice. When the record is the first in the slice, the previous position used is the slice alignment-start field (hence the first delta should be zero for single-reference slices, or the AP value itself for multi-reference slices). if **AP-Delta** = false: encodes the alignment start position directly
RG	encoding<int>	read groups	read groups. Special value '-1' stands for no group.
RN[a]	encoding<byte[ ]>	read names	read names
MF	encoding<int>	next mate bit flags	see specific section
NS	encoding<int>	next fragment reference sequence id	reference sequence ids for the next fragment
NP	encoding<int>	next mate alignment start	alignment positions for the next fragment
TS	encoding<int>	template size	template sizes
NF	encoding<int>	distance to next fragment	number of records to skip to the next fragment[b]
TL[c]	encoding<int>	tag ids	list of tag ids, see tag encoding section
FN	encoding<int>	number of read features	number of read features in each record
FC	encoding<byte>	read features codes	see separate section
FP	encoding<int>	in-read positions	positions of the read features; a positive delta to the last position (starting with zero)
DL	encoding<int>	deletion lengths	base-pair deletion lengths
BB	encoding<byte[ ]>	stretches of bases	bases
QQ	encoding<byte[ ]>	stretches of quality scores	quality scores
BS	encoding<byte>	base substitution codes	base substitution codes
IN	encoding<byte[ ]>	insertion	inserted bases
RS	encoding<int>	reference skip length	number of skipped bases for the 'N' read feature
PD	encoding<int>	padding	number of padded bases
HC	encoding<int>	hard clip	number of hard clipped bases
SC	encoding<byte[ ]>	soft clip	soft clipped bases
MQ	encoding<int>	mapping qualities	mapping quality scores
BA	encoding<byte>	bases	bases
QS	encoding<byte>	quality scores	quality scores
TC[d]	N/A	legacy field	to be ignored
TN[d]	N/A	legacy field	to be ignored

[a] Note RN this is decoded after MF if the record is detached from the mate and we are attempting to auto-generate read names.

[b] The count is reset for each slice so NF can only refer to a record later within this slice.

[c] TL is followed by decoding the tag values themselves, in order of appearance in the tag dictionary.

[d] TC and TN are legacy data series from CRAM 1.0. They have no function in CRAM 3.0 and should not be present. However some implementations do output them and decoders must silently skip these fields. It is illegal for TC and TN to contain any data values, although there may be empty blocks associated with them.

Note some auxiliary tags can be created automatically during decode so can optionally be removed by the encoder. However if the decoder finds a tag stored verbatim it should use this in preference to automatically computing the value.

The RG (read group) auxiliary tag should be created if the read group (RG data series) value is not -1.

The MD and NM auxiliary tags store the differences (an edit string) between the sequence and the reference along with the number of mismatches. These may optionally be created on-the-fly during reference-based sequence reconstruction and should match the description provided in the SAMtags document. An encoder may decide to store these verbatim when no reference is used or where the automatically constructed values differ to the input data.

Note there is no mechanism to describe which records have MD/NM present and which do not. If this is deemed important, the only recourse is to store all MD and NM verbatim and to request that the decoding software does not automatically generate its own for records that have no stored MD and NM tags.

10.6 Mapped reads

Read feature records

Read features are used to store read details that are expressed using read coordinates (e.g. base differences respective to the reference sequence). The read feature records start with the number of read features followed by the read features themselves. Each read feature has the position encoded as the distance since the last feature position, or the absolute position (i.e. delta vs zero) for the first feature. Finally the single mapping quality and per-base quality scores are stored.

Data series type	Data series name	Field	Description
int	FN	number of read features	the number of read features
int	FP	in-read-position[a]	~~position~~ delta-position of the read feature
byte	FC	read feature code[a]	See feature codes below
*	*	read feature data[a]	See feature codes below
int	MQ	mapping qualities	mapping quality score
byte[read length]	QS	quality scores	the base qualities, if preserved

[a] Repeated FN times, once for each read feature.

Read feature codes

Each feature code has its own associated data series containing further information specific to that feature. The following codes are used to distinguish variations in read coordinates:

This would be encoded as

binary	`01 10 00 11,`	`01 00 10 11,`	`10 00 01 11,`	`00 10 01 11,`	`00 01 10 11`
or hex	`0x63,`	`0x4b,`	`0x87,`	`0x27`	`0x1b.`

To decode, we would use the following lookup table, showing the same data as above with codes sorted into 0, 1, 2, 3 order.

	BS Code			
Ref. base	**0**	**1**	**2**	**3**
A	T	C	G	N
C	G	A	T	N
G	C	T	A	N
T	A	G	C	N
N	A	C	G	T

Substitution Code Assignment

There is no strict requirement on using a specific substitution matrix, nor that it be optimal. However one strategy may be to ensure the most common substitution is always given code 0, the next most common is code 1, and so on. This means the distribution of BS values will be skewed towards lower values, which helps improve compression over more uniformly distributed frequencies.

For example, let us assume the following substitution frequencies for base A:

AC: 15%
AG: 25%
AT: 55%
AN: 5%

Then the substitution codes are T=0, G=1, C=2, N=3.

Decode mapped read pseudocode

```
1:  procedure DecodeMappedRead
2:      feature_number ← ReadItem(FN, Integer)
3:      last_feature_position ← 0
4:      for i ← 1 to feature_number do
5:          DecodeFeature
6:      end for
7:      mapping_quality ← ReadItem(MQ, Integer)
8:      if CF AND 1 then                          ▷ Quality stored as an array
9:          for i ← 1 to read_length do
10:             quality_score ← ReadItem(QS, Integer)
11:         end for
12:     end if
13: end procedure

14: procedure DecodeFeature
15:     feature_code     ← ReadItem(FC, Integer)
16:     feature_position ← ReadItem(FP, Integer) + last_feature_position
17:     last_feature_position ← feature_position
18:     if feature_code ='B' then
19:         base             ← ReadItem(BA, Byte)
20:         quality_score    ← ReadItem(QS, Byte)
21:     else if feature_code ='X' then
22:         substitution_code ← ReadItem(BS, Byte)
23:     else if feature_code ='I' then
24:         inserted_bases   ← ReadItem(IN, Byte[])
25:     else if feature_code ='S' then
26:         softclip_bases   ← ReadItem(SC, Byte[])
```