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 (1-based)
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 (1-based)
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.

Bit flag	Name	Description
0x1	quality scores stored as array	quality scores can be stored as read features or as an array similar to read bases.
0x2	detached	mate information is stored verbatim (e.g. because the pair spans multiple slices or the fields differ to the CRAM computed method)
0x4	has mate downstream	tells if the next segment should be expected further in the stream
0x8	decode sequence as "*"	informs the decoder that the sequence is unknown and that any encoded reference differences are present only to recreate the CIGAR string.

The following pseudocode describes the general process of decoding an entire CRAM record. The sequence data itself is in one of two encoding formats depending on whether the record is aligned (mapped).

Decode pseudocode

```
procedure DecodeRecord
    $BAM\_flags$ ← ReadItem(BF, Integer)
    $CRAM\_flags$ ← ReadItem(CF, Integer)
    DecodePositions                              ▷ See section 10.2
    DecodeNames                                  ▷ See section 10.3
    DecodeMateData                               ▷ See section 10.4
    DecodeTagData                                ▷ See section 10.5

    if ($BF$ AND 4) = 0 then                     ▷ Unmapped flag
        DecodeMappedRead                         ▷ See section 10.6
    else
        DecodeUnmappedRead                       ▷ See section 10.7
    end if
end procedure
```

This pseudocode is not meant to be a fully implementable programming language, but to act as an algorithmic guide to the order and structure of CRAM decoding.

The ReadItem function referred above takes two arguments; the data series name and the data type used by the Encoding. It will use the codec specified in the Container Compression Header to retrieve the next value from that data series. Note there is only one permitted data type per data series, so the second argument is redundant and is included only as an aide-mémoire.

10.2 CRAM positional data

Following the bit-wise BAM and CRAM flags, CRAM encodes positional related data including reference, alignment positions and length, and read-group. Positional data is stored for both mapped and unmapped sequences, as unmapped data may still be "placed" at a specific location in the genome (without being aligned). Typically this is done to keep a sequence pair (paired-end or mate-pair sequencing libraries) together when one of the pair aligns and the other does not.

For reads stored in a position-sorted slice, the AP-delta flag in the compression header preservation map should be set and the AP data series will be delta encoded, using the slice alignment-start value as the first position to delta against. Note for multi-reference slices this may mean that the AP series includes negative values, such as when moving from an alignment to the end of one reference sequence to the start of the next or to unmapped unplaced data. When the AP-delta flag is not set the AP data series is stored as a normal integer value, using 1-based coordinates as per SAM.

(mapped status, orientation) for both records. In this case both records are labelled as “detached” in the CF data series using bit 2.

If this and the next fragment are within the same slice, we can derive much of this information by comparing the two records. The upstream record has CF bit 4 (mate downstream) flag set and stores the number of records to skip (in the NF data series) between this record and the record for the next fragment on this template, with zero meaning the next fragment is also the next record. The downstream record has neither CF bits 2 (detached) or 4 (mate downstream) set nor does it use the NF data series (unless it also has an additional “next fragment” to refer to).

It is not mandatory to use this deduplication approach and optionally CRAM write implementations may wish to label data as detached even when all records for the template reside in the same slice. One reason to do this may be to preserve inconsistent data so that it round-trips through the CRAM format with full fidelity

Data series type	**Data series name**	**Description**
int	NF	the number of records to skip to the next fragment

In the above case, the NS (mate reference name), NP (mate position) and TS (template size) fields for both records should be derived once the mate has also been decoded. Mate reference name and position are obvious and simply copied from the mate. The template size is computed using the method described in the SAM specification; the inclusive distance from the leftmost to rightmost mapped bases with the sign being positive for the leftmost record and negative for the rightmost record.

If the next fragment is not found within this slice then the following structure is included into the CRAM record. Note there are cases where read-pairs within the same slice may be marked as detached and use this structure, such as to store mate-pair information that does not match the algorithm used by CRAM for computing the mate data on-the-fly.

Data series type	**Data series name**	**Description**
int	MF	next mate bit flags, see table below
byte[ ]	RN	the read name (if and only if not known already)
int	NS	mate reference sequence identifier
int	NP	mate alignment start position (1-based)
int	TS	the size of the template (insert size)

Next mate bit flags (MF data series)

The next mate bit flags expressed as an integer represent the MF data series. These represent the missing bits we excluded from the BF data series (when compared to the full SAM/BAM flags). The following bit flags are defined:

Bit flag	**Name**	**Description**
0x1	mate negative strand bit	the bit is set if the mate is on the negative strand
0x2	mate unmapped bit	the bit is set if the mate is unmapped

Decode mate pseudocode

In the following pseudocode we are assuming the current record is *this* and its mate is *next_frag*.

```
procedure DecodeMateData
    if $CF$ AND 2 then                                          ▷ Detached from mate
        $mate\_flags \leftarrow$ ReadItem(MF,Integer)
        if $mate\_flags$ AND 1 then
            $bam\_flags \leftarrow bam\_flags$ OR 0x20                ▷ Mate is reverse-complemented
        end if
        if $mate\_flags$ AND 2 then
            $bam\_flags \leftarrow bam\_flags$ OR 0x08                ▷ Mate is unmapped
        end if
        if $container\_pmap.read\_names\_included \neq 1$ then
            $read\_name \leftarrow$ ReadItem(RN, Byte[])
```