Encoding notation is defined as the keyword ‘encoding’ followed by its data type in angular brackets, for example ‘encoding<byte>’ stands for an encoding that operates on a data series of data type ‘byte’.

Encodings may have parameters of different data types, for example the EXTERNAL encoding has only one parameter, integer id of the external block. The following encodings are defined:

Codec	**ID**	**Parameters**	**Comment**
NULL	0	none	series not preserved
EXTERNAL	1	int block content id	the block content identifier used to associate external data blocks with data series
Deprecated (GOLOMB)	2	int offset, int M	Golomb coding
HUFFMAN	3	array<int>, array<int>	coding with int/byte values
BYTE_ARRAY_LEN	4	encoding<int> array length, encoding<byte> bytes	coding of byte arrays with array length
BYTE_ARRAY_STOP	5	byte stop, int external block content id	coding of byte arrays with a stop value
BETA	6	int offset, int number of bits	binary coding
SUBEXP	7	int offset, int K	subexponential coding
Deprecated (GOLOMB_RICE)	8	int offset, int $\log_2 m$	Golomb-Rice coding
GAMMA	9	int offset	Elias gamma coding

See section 13 for more detailed descriptions of all the above coding algorithms and their parameters.

4 Checksums

The checksumming is used to ensure data integrity. The following checksumming algorithms are used in CRAM.

4.1 CRC32

This is a cyclic redundancy checksum 32-bit long with the polynomial 0x04C11DB7. Please refer to ITU-T V.42 for more details. The value of the CRC32 hash function is written as an integer.

4.2 CRC32 sum

CRC32 sum is a combination of CRC32 values by summing up all individual CRC32 values modulo 2^{32}.

5 File structure

The overall CRAM file structure is described in this section. Please refer to other sections of this document for more detailed information.

A CRAM file consists of a fixed length file definition, followed by a CRAM header container, then zero or more data containers, and finally a special end-of-file container.

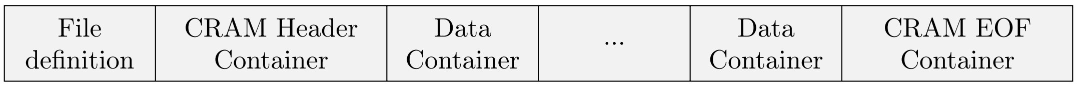

Figure 1: A CRAM file consists of a file definition, followed by a header container, then other containers.

Containers consist of one or more blocks. The first container, called the CRAM header container, is used to store a textual header as described in the SAM specification (see the section 7.1). This container may have additional padding bytes present for purposes of permitting inline rewriting of the SAM header with small changes in size. These padding bytes are undefined, but we recommend filling with nuls. The padding bytes can either be in explicit uncompressed Block structures, or as unallocated extra space where the size of the container is larger than the combined size of blocks held within it.

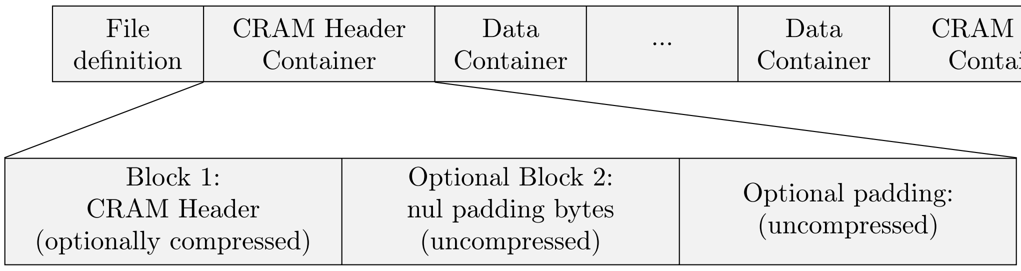
File definition
CRAM Header Container
Data Container
...
Data Container
CRAM EOF Container
Block 1: CRAM Header (optionally compressed)
Optional Block 2: nul padding bytes (uncompressed)
Optional padding: (uncompressed)

Figure 2: The the first container holds the CRAM header text.

Each container starts with a container header structure followed by one or more blocks. The first block in each container is the compression header block giving details of how to decode data in subsequent blocks. Each block starts with a block header structure followed by the block data.

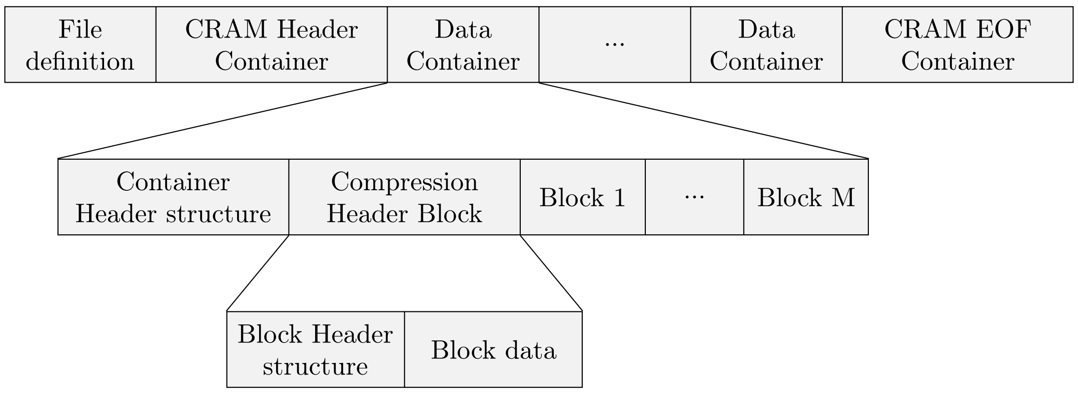

Figure 3: Containers as a series of blocks

The blocks after the compression header are organised logically into slices. One slice may contain, for example, a contiguous region of alignment data. Slices begin with a slice header block and are followed by one or more data blocks. It is these data blocks which hold the primary bulk of CRAM data. The data blocks are further subdivided into a core data block and one or more external data blocks.

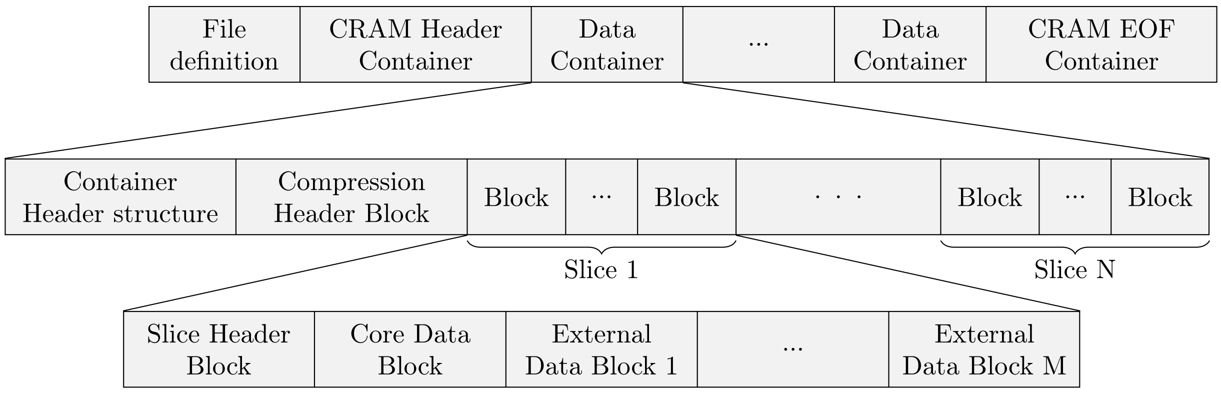

Figure 4: Slices formed from a series of concatenated blocks

6 File definition

Each CRAM file starts with a fixed length (26 bytes) definition with the following fields:

Data type	**Name**	**Value**
byte[4]	format magic number	CRAM (0x43 0x52 0x41 0x4d)
unsigned byte	major format number	3 (0x3)
unsigned byte	minor format number	1 (0x1)
byte[20]	file id	CRAM file identifier (e.g. file name or SHA1 checksum)

Valid CRAM *major.minor* version numbers are as follows:

1.0 The original public CRAM release.

2.0 The first CRAM release implemented in both Java and C; tidied up implementation vs specification differences in *1.0*.

2.1 Gained end of file markers; compatible with *2.0*.

3.0 Additional compression methods; header and data checksums; improvements for unsorted data.

3.1 Additional EXTERNAL compression codecs only.

CRAM 3.0 and 3.1 differ only in the list of compression methods available, so tools that output CRAM 3 without using any 3.1 codecs should write the header to indicate 3.0 in order to permit maximum compatibility.

7 Container header structure

The file definition is followed by one or more containers with the following header structure where the container content is stored in the 'blocks' field:

Data type	**Name**	**Value**
int32	length	the sum of the lengths of all blocks in this container (headers and data) and any padding bytes; equal to the total byte length of the container minus the byte length of this header structure
itf8	reference sequence id	reference sequence identifier or -1 for unmapped reads -2 for multiple reference sequences. All slices in this container must have a reference sequence id matching this value.
itf8	starting position on the reference	the alignment start position
itf8	alignment span	the length of the alignment
itf8	number of records	number of records in the container
ltf8	record counter	1-based sequential index of records in the file/stream.
ltf8	bases	number of read bases
itf8	number of blocks	the total number of blocks in this container
array<itf8>	landmarks	the locations of slices in this container as byte offsets from the end of this container header, used for random access indexing. For sequence data containers, the landmark count must equal the slice count. Since the block before the first slice is the compression header, landmarks[0] is equal to the byte length of the compression header.
int	crc32	CRC32 hash of the all the preceding bytes in the container.
byte[ ]	blocks	The blocks contained within the container.

In the initial CRAM header container, the reference sequence id, starting position on the reference, and alignment span fields must be ignored when reading. The landmarks array is optional for the CRAM header, but if it exists it should point to block offsets instead of slices, with the first block containing the textual header.

In data containers specifying unmapped reads or multiple reference sequences (i.e. reference sequence id < 0), the starting position on the reference and alignment span fields must be ignored when reading. When writing, it is recommended to set each of these ignored fields to the value 0.

7.1 CRAM header container

The first container in a CRAM file contains a textual header in one or more blocks. See section 8.3 for more details on the layout of data within these blocks and constraints applied to the contents of the SAM header.