**INTERNATIONAL ORGANISATION FOR STANDARDISATION**
**ORGANISATION INTERNATIONALE DE NORMALISATION**
**ISO/IEC JTC1/SC29/WG11**
**CODING OF MOVING PICTURES AND AUDIO**

**ISO/IEC JTC1/SC29/WG11 M45601**
**January 2019, Marrakesh, MA**

| | |
|---|---|
| **Source** | **Requirements** |
| **Status** | **Input document** |
| **Title** | **Comments on ISO/IEC DIS 23092-1 and ISO/IEC DIS 23092-2** |
| **Author** | Daniel Naro (UPC), Jan Voges (Leibniz University Hannover), Tom Paridaens (UGent-imec), Dmitry Repchevski (BSC), Jaime Delgado (UPC), Paolo Ribeca (The Pirbright Institute), Idoia Ochoa (UIUC), Mikel Hernaez (UIUC) |

## Table of contents



# 1 Purpose

This document summarizes comments on ISO/IEC DIS 23092-1 and ISO/IEC DIS 23092-2.

# 2 Comments on the class concept

In the DIS of Part 1 and Part 2 under ballot the concept of classes plays two – almost orthogonal – roles. The first role is related to random access: by specifying a class of data one can

better target which sort of information will be retrieved. The second role is related to decoding process selection: depending on the class, the decoding process is changed, and thus also the requirements. Let us discuss the suitability of classes to address these two roles.

However, first, we need to address the functionalities covered by the descriptors rftp and rftt. The descriptors rftp and rftt are meant to store all differences with respect to the reference which are shared by a large number of reads. Let us consider an example where a sample exhibits only homozygous mutations (SNPs and/or indels) and where the sequencing technology generates no mistakes: to achieve a reasonable compression performance and encoder will represent all mutations by using the descriptors rtfp and rftt, and the descriptors mmpos and mmtype will not be used. It is not clear how to classify these data. If the class is to be relevant for random access, the class should be I, as there are mutations of any type. Nevertheless, this is detrimental for selecting the least consuming decoding process. If the class is to be informative for the decoding process the class should be P, and this is detrimental to the random access feature.

## *2.1 Decoding process selection*

### 2.1.1 Problem

The APIs, specified in Part 3, are the way to query the information represented as specified in Part 1. In the workflow, the user queries the information through the APIs. The application decides which elements from Part 1 need to be accessed based on the requested method. The results of API calls are ultimately Data Units to be submitted to the decoder (i.e., Part 2). If more filtering is necessary, the API calls are lacking functionality, or the format is lacking functionality. These issues need to be addressed in Part 1 or Part 3.
The role that classes do play in Part 2 is to specify which decoding process is to be executed. We believe that there is no need for this selection, in the case of aligned data:

- Class P is equal to class N where for each read we obtain that there are no SNPs.
- Class N is equal to class M where for each SNP we retrieve that it is a flip to N.
- Class M is equal to class I, where we never retrieve any clip.

This shows that there is only one decoding process, namely the one for class I. All other classes are special cases, where due to the absence of certain descriptor streams a default behaviour is defined for each of them.

### 2.1.2 Solution

An encoder is free to use these default behaviours to do exactly what a current encoder would be doing. As a matter of fact, thanks to these default behaviours, we could envision new partitions: for example, an Access Unit (AU) with no further changes than soft clips (i.e., an AU of class P with clips). Note that there is no need to create a new class for this.

### 2.1.3 Consequences of the change

There are no changes in the Data Units to be decoded: the bitstreams will be the same. This change does not require a new Core Experiment: the semantics remain untouched.
The content of an AU can easily be scanned to discover which resources will be needed: this can be achieved by just checking which Descriptors are present and by deriving from this whether, for example, a new (CABAC) parser is required.
Furthermore, currently the class of the AU is not enough information to derive the amount of resources required. On the one hand, the class is not a reliable metric to derive the number of required CABAC decoders, as there are optional descriptor streams. On the other hand, classes are not a way to filter which operation will have to be applied during the decoding process. As we

have shown, there might be mutations, insertions or deletions in the output of an AU of any class due to the rftp and rftt descriptor streams.

## *2.2 Random access*

### 2.2.1 Problem

Let us focus our attention to the other key role that classes play. Classes allow us to target sets of reads based on certain characteristics. The working group had positive feedback from the research community regarding this feature.
However, using the AU type as a way to convey characteristics about its content is detrimental to the selection of the decoding process.

### 2.2.2 Solution

Currently there are the classes P, N, M, I, HM, U. We envision the following alternative approach: the Master Index Table (MIT) holds part of the metadata as prefetched content (similarly to how au_end_position is stored), and if required, the application will be able to discover more in the Part 3 structures.
To implement the current functionalities, we would only need four fields:

- average_clip_size: if this field is not equal to zero this is equivalent to indicating class I
- average_indel_size: if this field is not equal to zero this is equivalent to indicating class I
- average_number_of_mismatches: if this is not equal to zero this is equivalent to indicating another class than P
- only_n_mismatches: flag discriminating classes N and M, when average_clip_size is equal to 0

Functionality to retain the class identifiers P, N and M can be modified accordingly. For example, class M can be identified with average_clip_size set to 0, average_indel_size set to 0, average_number_of_mismatches set to something else than 0.

### 2.2.3 Consequences of the change

Specifying the mentioned fields in the MIT and in Part 3 ensures that all filtering capabilities are maintained. What is more, the filtering is augmented and performed in the “right spot”.
We should highlight that this approach uses two mechanisms which are already present in the standard. Given the current definition, the random access mechanism requires information from other sources than the MIT to perform a sounded access. For example, we need to know the threshold at which read pairs are split, information which is not stored in the MIT. Therefore, arguing against removing information which is needed for random access from the MIT does not take into account that some of the information needed for random access is already outside the MIT. Similarly, the MIT already includes information which is not mandatory but helpful in the form of the AU end position.

# 3 Comments on the current mmtype and rftt specification

This section first lists problems with the current specification of mmtype and rftt which need to be addressed. In a second part a possible solution is proposed.

## *3.1 Problems*

### 3.1.1 Problem with mmtype

#### 3.1.1.1 Issue with multiple mmtype symbols of SNP type for a same position

Nothing in the current specification avoids the case where multiple mmtype symbols of SNP type for a same position are given. In the case of the decoding of the sequence this is not a real issue: by applying the decoding process properly, the correct sequence is obtained.
The problem resides in the decoding of the extended CIGAR string (Table 1. Decoding process for the ecigar_string field in the MPEG-G record.). In the case where this contradictory information is provided, the decoding process will generate wrong data. Upon reaching the first mutation signaling, the extended CIGAR string is concatenated with the mutated value. Then, when reaching the second mutation (which is for the same position), the extended CIGAR string is again concatenated, when in fact the last value should be replaced.

#### 3.1.1.2 Issue with combination of the mmtype deletion symbol and other mmtype symbols

As currently defined a mmtype delete symbol could be used to delete a previously introduced or changed symbol. Again, the decoding of the sequence is correct, but the decoding of the extended CIGAR string is not, as no filtering for edge cases is performed, thus leading to an uncontrolled concatenation of operations. In other words, the output contains CIGAR operations for elements which are not present in the sequence.
There are multiple possible solutions to address these issues:

- fix the decoding process of the extended CIGAR string by making it considerably more complex
- declare some combinations of mmtype/mmpos as prohibited or with undefined behavior
- modify how mmtype is specified to make these cases impossible

### 3.1.2 Problem with rftp

The current implementation of the functionality of reference modification through rftp and rftt seems to be not answering a real use case.
If the data to be encoded is aligned to a slightly different reference, one can use the functionalities of MPEG-G to define a reference based on another reference. In this case it is understood that the reference is changed; therefore, also the alignment is changed: for example, what with the original reference appeared to be an insertion might now be a match or a mutation.
Considering that this functionality is already covered, rftp and rftt should allow to codify mutations present in all reads of the AU (for that position), and the mutation should appear as a mutation in the read's extended CIGAR string. However, currently the mutation is not reflected in the CIGAR string. The user will need to compare the output of different AUs and possibly the reference, to detect that some CIGAR strings are inconsistent. Furthermore, if some change introduced by rftt needs to be reverted for one position, the modification will appear as mutation, although it is in fact indicating that for that position the read is the same as the reference.

## *3.2 Proposed solution*

The solution revolves around two ground ideas. First, only one modification per position might exist. Second, the solution works with position on the reference, not position on the read being modified. This allows merging of information coming from rftt and mmtype, and makes it easy to implement an algorithm which does not require memory shifting.
The decoding process would be as follows. For each position of the reference visited by the decoding process, we first check in rftt and then in mmtype if a modification is known (if known

in both sources, the second one is preferred), if such is the case, the sequence and the extended CIGAR string are updated accordingly. A modification is represented as follows:

- modification type: delete, SNP_with_possible_insert, insert_only
- length_of_insert: field not provided for delete
- insert alphabet symbols: a list of size length_of_insert, containing all alphabet symbols being inserted (in between the prior and current position)

This approach makes sure that no deletion can be applied to something which has just been modified. No operation can be applied twice, for example position 5 modified to C and then to G. This approach might change somewhat the result obtained in case of a round trip: a deletion and then insertion is necessarily changed to a SNP (which anyways makes more sense). Finally, special attention has to be given to the special case where the very last operation prior to a possible soft-clip is an insertion.

# 4 Comments on multiple alignments

## *4.1 Concept of primary alignment*

The concept of "primary alignment" is artificial. Although each sequencing read does originate from one specific location in the genome, in general it is perfectly possible for it to align equally well to more than one single genomic location – that happens whenever the sequence of the location the reads originates from is repeated many times in the genome. In such a case it is not possible for an aligner to determine which copy of the sequence in the genome is the "correct" one, and it *must* report *all* the equivalent alignments found as equally probable. Historically most early-day mappers artificially chose to report one "primary" match (sometimes selecting one of the equivalent alignments at random) and this arbitrary design choice eventually crept into SAM; however, such an approach is completely inadequate to many realistic biological situations, such as mapping to repetitive regions of the genome or aligning RNA-sequencing data.
Hence, we suggest discussing the concept of "primary" alignment.

## *4.2 Identification of secondary alignments*

In Part 2, there is no mechanism to signal that a record is a secondary alignment in the case where the mscore descriptor is missing. Thus, all alignments of the read should be retrieved in order to discover which one was the primary alignment.
In the decoding process of the pair descriptor there is the special case marked as "more_align", but this indicates only the position of the other alignment.
It should also be noted that two reads might be mapped to the same position, and in that case the proposed syntax for "more_align" in the pair descriptor is not able to discriminate efficiently which one is the other alignment. Mitigation solutions might be to use the read name or to compare the sequence, but these solutions are not specified in the standard.
Furthermore, when decoding the alignments of the right read, the first step is to know how many alignments there are for the given alignment of the left read. A special case is introduced to signal that the alignment of the right read is an already known alignment, by indicating that there are "0" alignments of the right read attached to the alignment of the left read. The mechanism should be clarified on how it handles these cases:

- What happens if there are $N_{total} = N_{known} + N_{unknown}$ alignments for the right read, both $N_{known}$, $N_{unknown}$ being greater or equal to one? Currently it appears as if the standard only handles $N_{known} >= 1$ and $N_{unknown} = 0$, and $N_{known} = 0$ and $N_{unknown} = 1$.
- What value should be given to indicate that the alignment of the right read is unmapped? Should the number of alignments not be 0 then? But this contradicts the special value's meaning.