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

ISO/IEC JTC1/SC29/WG11 MPEG2020/M54565
June 2020, Online

Source: ISO/IEC JTC1/SC29/WG11
Status: Input Document
Title: **Proposed Enhancements to the Working Draft of MPEG-G Part 6**
Author: Patrick Y.H. Cheung (Royal Philips), Shubham Chandak (Stanford University), Mikel Hernaez (Center for Applied Medical Research at University of Navarra, UIUC), Idoia Ochoa (Tecnun at University of Navarra, UIUC)

1 Introduction

The goal of this document is to propose different ways for enhancing the functionalities and features of the current working draft for Part 6 (N19314) based on the ideas from the various core experiment proposals submitted for the MPEG 130 meeting. The values of these proposed enhancements are further illustrated through the supporting use cases described in the companion document M54566.

2 Multi-dimensional Data

In the requirements document for Part 6 (N18647), the support for a compressed representation (Req ID 4.6.1) and an efficient encoding (Ref ID 4.6.3 and 4.2.4) of multidimensional numerical values is required for Hi-C and quantitative browser track (wig, bigWig or bedGraph) data. The idea is to represent a collection of matrices as a multi-dimensional array. While the original purpose is to allow Hi-C or quantitative browser track data to be stored at different resolutions, the multi-dimensional structure is also useful for representing time-series gene expression and other quantitative data. Specifically for handling multiple resolutions, precomputing the data at different resolutions and storing it in distinct tables allows rapid visualization at any zoom/resolution. It also enables using the appropriate chunk size (in terms of genomic range) for the specific resolution level rather than forcing the same size across resolutions which is highly inefficient for the coarser resolutions. The methods, challenges and opportunities of the analysis of time-series gene expression data are summarized in [1].

To enable this feature in the current working draft for Part 6, we propose to introduce an additional level of *Annotation Table* container within the Dataset container, where each Annotation Table corresponds to a specific resolution or time-point. Information on the resolution or time-point can then be specified in the Annotation Table Header. Figure 1 shows how multi-resolution or time-series data can be organized in multiple Annotation Tables within the same dataset. Besides the representation of multi-dimensional data, the additional container is also beneficial for providing another level of data grouping, which is desirable for improving the flexibility of data organization in large genomic studies.

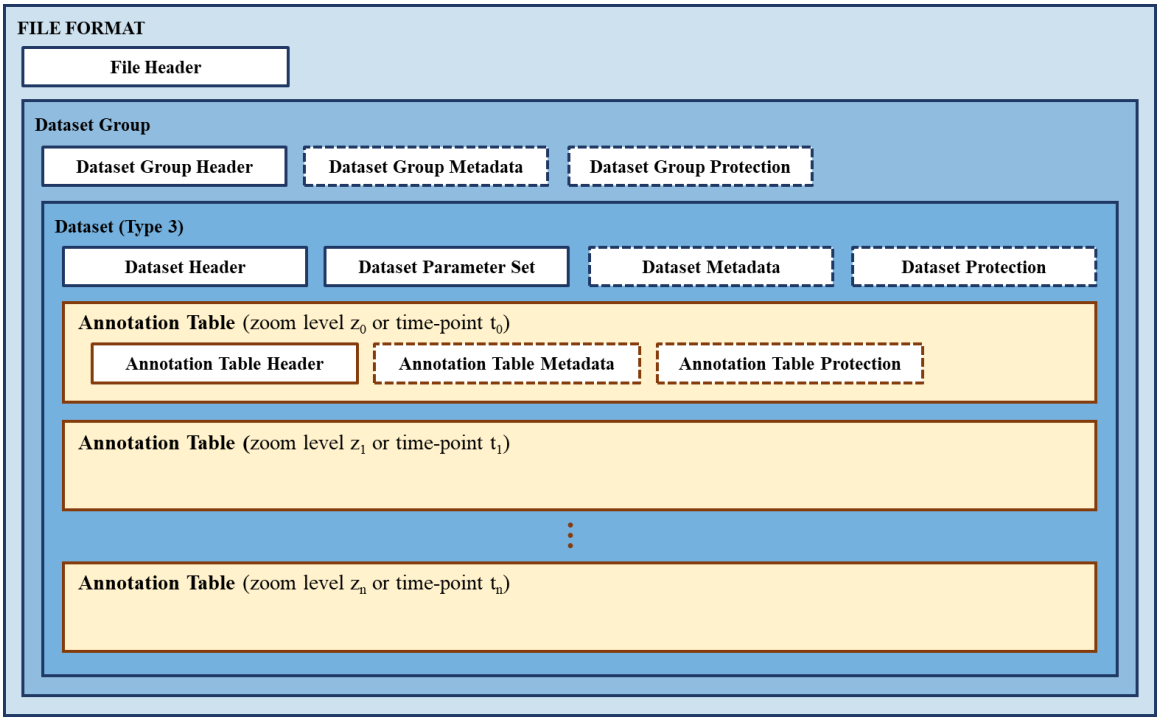
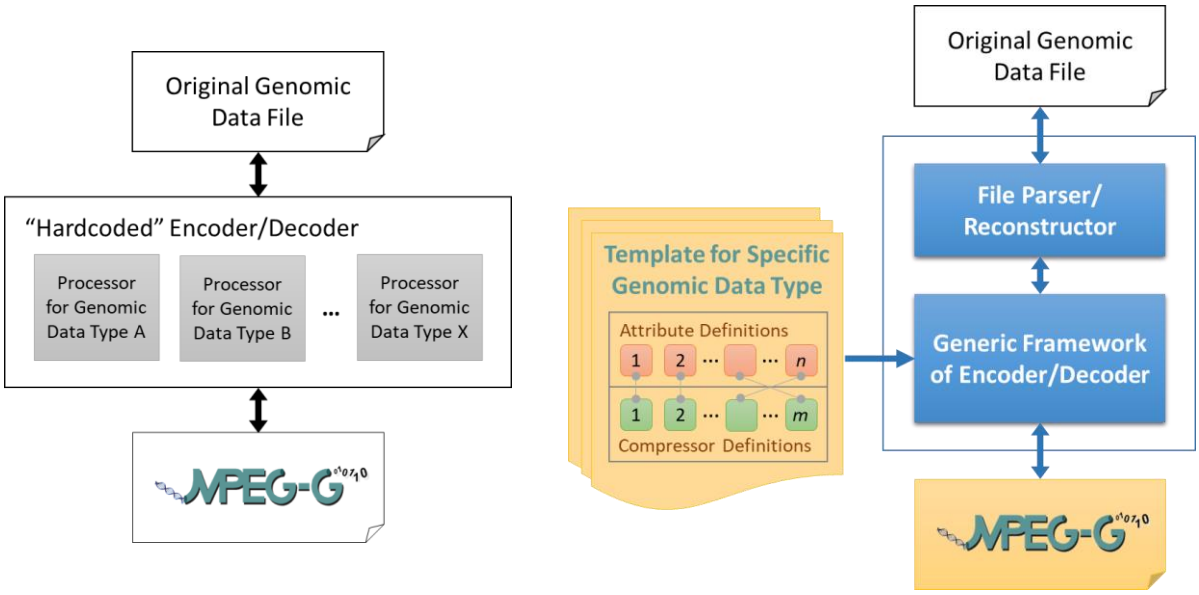


Figure 1 – Use of Annotation Table containers for representing data at multiple resolutions or time points

3 Uniform Interface for Defining Attributes and Compressors

In the fast-changing field of genomics, it is common for existing data formats to be modified and new data formats created at the emergence of new bioinformatics tools and biotechnologies. A successful standard for genomic data should therefore be able to adapt to these changes in a timely manner by minimizing the time and efforts required for updating the syntax and reference software. One way for achieving this goal is to provide a uniform interface for the definition of attributes and their corresponding compressors. With this approach, a genomic data format can be fully specified by a template of attribute and compressor definitions using the uniform interface. To support a new format, only the template needs to be customized, while the generic framework for template interpretation and data processing can remain the same. In this way, an update to the genomic data standard for a new file format simply involves adding a customized template to the specifications, without the need to make any further changes to the core syntax or reference software. Figure 2 shows a comparison between the architectures of (a) hardcoded processing of each genomic data type as described in the working draft, and (b) customizable processing that ingests the template of attributes and compressors defined for each genomic data type through a uniform interface.



(a) (b)

Figure 2 – The software architectures of (a) hardcoded processing, and (b) customizable processing that ingests the template of attribute and compressor definitions for each genomic data type

3.1 Attributes

In the current working draft, attributes/descriptors are divided into two broad categories: main (common/specific) and configurable. While the configurable attributes are defined using a uniform interface as specified in subclause 9.3 of the working draft, there is a lack of uniform interface for defining the main attributes and their parameters. In fact, the main attributes are only defined in the text of the specification, but not explicitly defined in a structured template, thus strictly limiting the flexibility for making adjustments to the attributes, such as their data types and compression algorithms, and requiring the processing steps to be hardcoded in the software. We therefore propose applying the same interface for defining the configurable attributes to all attributes. Table 1 shows the syntax of an *attribute_information* structure, which is a collection of attribute definitions encapsulated in *attribute_parameter_set*, with the number of attributes specified in n_attributes.

Table 1 – Table Data Attribute Information syntax

Syntax	**Key**	**Type**
attribute_information {	*tdai*	
n_attributes		u(16)
for (i=0; i<n_attributes; i++)		
attribute_parameter_set[i]	tdap	gen_info
}		

3.2 Compressors

We propose a unified interface for defining the compressor (transforms + entropy coding) across attributes, both required/core/default attributes and optional/dynamic attributes. Thus, rather than hardcoding the subsequences and compressors for individual attributes as part of the standard (as done in input document M52990 or in working draft section 8.3.1.2.2.1), flexibility should be

provided to the user in this regard. We propose that in most cases, the specialized subsequences and compressors for certain attributes (e.g., alternative alleles) should be included as distinct compressors/transforms that can be optionally applied by the user. This proposal while incorporating GPress [2] enables such a mode, wherein specialized transforms can be applied to certain attributes, but only if the user wishes to (e.g., see the results in input document M52159 clause 4). Otherwise, the user can choose to directly apply the entropy coder to the attribute. We propose that a hardcoded representation in terms of subsequences should be used only if it is semantically meaningful and part of the decoded record. Even in such cases, flexibility should be provided with regards to the compressor to be used on these subsequences. The details of such an approach are discussed in input document M53381 subclause 6.4.13 (also copied below in Table 2). We list some advantages of this approach below:

- Simplified interface uniformly applicable to all attributes.
- In many cases, splitting the attributes into multiple subsequences only increases the processing time without providing appreciable benefits over simply treating the attribute as a null-terminated string and applying a universal compressor such as BSC. Examples include fields such as alternate allele (ALT) and genotype (GT) in the VCF file.
- In some cases, splitting an attribute into multiple subsequences can be disadvantageous if they are compressed individually without utilizing potential dependencies between them.
- The proposal still supports use of specialized compressors in cases where they provide benefits, without enforcing their use in all circumstances.

The proposed framework can be incorporated into the working draft section 7.11 (parameter set).

Table 2 – Compressor syntax

Syntax	**Type**
compressor {	
compressor_ID	st(v)
reserved	u(7)
transform	u(1)
if (transform) {	
transform_algorithm_ID	st(v)
n_dependencies	u(8)
}	
n_compression_algorithms	
for (i=0; i<n_compression_algorithms; i++) {	
compression_algorithm_ID[i]	st(v)
compression_algorithm_pars[i]	st(v)
}	
}	

4 Attribute Grouping

We propose the incorporation of auxiliary attributes representing the attributes specific to the rows or columns for two-dimensional datasets (VCF with samples, gene expression), rather than storing them as separate datasets (as done in input document M52990 clause 5). For example, in a VCF file with samples, the variant information (chromosome, position, REF, ALT, etc.) are the row-specific attributes while the sample name is a column-specific attribute. The corresponding genotype is the main two-dimensional table in this scenario. In most use cases, the genotypes,

variants and samples are specific to a particular file with little possibility of reusing the same variant information across files. Moreover, the typical use case involves accessing all three classes of attributes (genotypes, samples, variants) since the genotype by itself does not convey useful information. Hence storing them as separate datasets does not provide an advantage in most cases, instead complicated the decoding procedure since the relevant information is spread across datasets.

Instead this proposal groups the attributes into those that correspond to the main two-dimensional table and those that correspond to the rows/columns. These attributes are still stored separately to allow efficient compression and random access but are kept within a single table structure. One advantage of this approach is the straightforward procedure to query the main table attributes (e.g., genotypes) based on the values of the auxiliary row attributes (e.g., variant position) by using a single index mapping the genomic position to the rows of the table. Another advantage is the unified interface to support different two-dimensional datatypes such as VCF and gene expression, without the need to explicitly define the link between different dataset types. See figure below for illustrations of this approach and an application to VCF files.

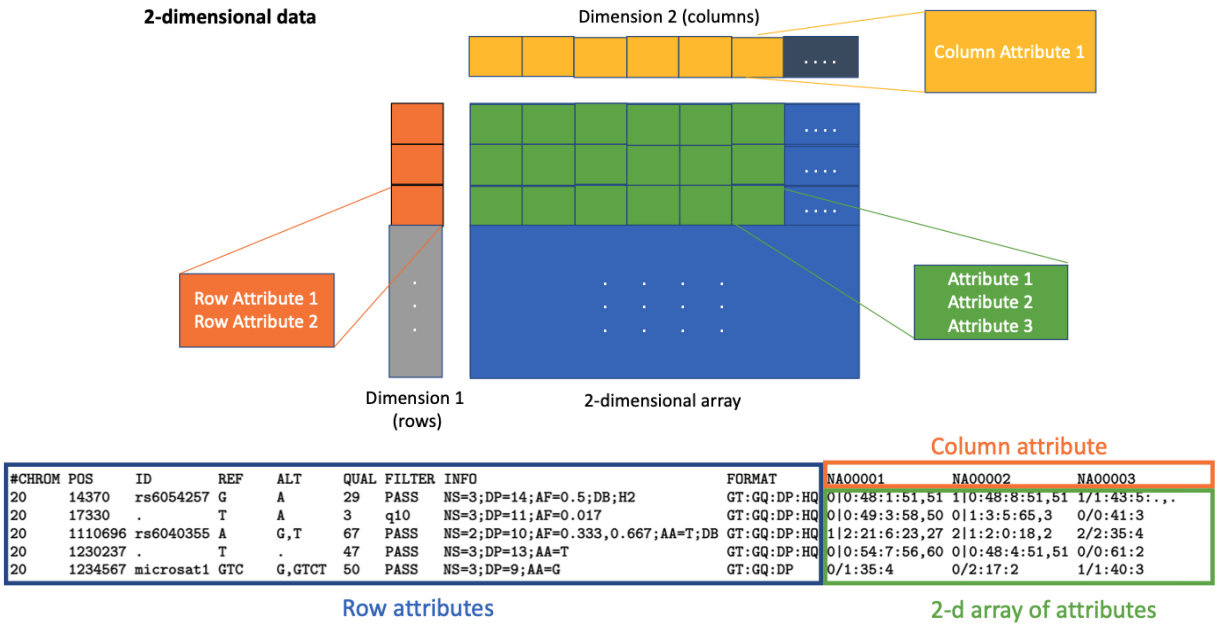


Figure 3: Illustration of 2-dimensional data with multiple attributes. The 2-dimensional data contains dimension-specific attributes in addition to the main 2-dimensional array. Also shown is an example application of this approach on a VCF file.

This proposal also supports additional auxiliary attribute groups to support linkage attributes, allowing for logical separation of data attributes and linkage attributes. As an example, linkage attributes (column) can be used to store the dataset group id with the sequencing data corresponding to each sample. This provides an elegant mechanism for linking different datasets, and can be easily merged with a proposal which supports metadata and UUID for the samples.

The concept and syntax of these auxiliary attributes is detailed in input document M53381 subclause 6.4.3.3, examples of these concept applied to the relevant file formats is detailed in input document M53383 clause 3, and results for decoding and random access using this mechanism are in input document M52159 clause 4.

5 Flexible Payload Organization and Ordering

There is a need to support different modes of organizing and ordering the compressed payloads depending on the specific requirements of an application. The following are some examples.

- If data is expected to be more frequently accessed in bulk from individual attributes, then *attribute contiguity*, i.e. grouping payloads of all chunks (blocks of rows and columns in an annotation table) belonging to the same attribute into an access unit, should be applied. This could be the case, for example, when one only requires access to the genotype (GT) field in a VCF file and not the remaining fields like genotype likelihood (GL).
- If data is expected to be more frequently accessed from multiple rows across all attributes, then *chunk contiguity*, i.e. grouping payloads of all attributes belonging to the same chunk, should be applied. This could be the case, for example, when one requires access to all the fields in a VCF file belonging to a specific genomic region.
- For two-dimensional attributes, if data is usually accessed in columns (or rows), then *column-major (or row-major) ordering* of the chunk payloads should be applied. This should be set based on the common access patterns of the data and can improve the access speed due to disk caching.

To support flexible payload organization and ordering, we propose to add *attribute_contiguity* and *column_major_chunk_order* flags to the Annotation Table Header or other appropriate data structures.

6 Data Symmetry Modes

If a two-dimensional attribute is expected to hold symmetrical data, then the storage size can be reduced by removing the redundant data in the symmetrical half. This can be done if the symmetry mode of the attribute is properly specified. This is particularly relevant for contact matrices (Hi-C) which are symmetric by definition, and allows representation of this data in the unified two-dimensional attribute interface without needing to define a new data structure. For this purpose, we propose to add *symmetry_mode* and *symmetry_minor_diagonal* fields to the Annotation Table Header or other appropriate data structures:

- **symmetry_mode** specifies the symmetry mode of the main Table Data and is only effective when two_dimensional_main == 1. The possible values are: 0 = unsymmetrical; 1 = symmetrical; 2 = skew-symmetric; 3 = Hermitian; 4-7 = reserved or user-defined. For symmetry modes 1-3, attribute values in the reflected half to the right of the principal/minor diagonal (inclusive of the diagonal if skew-symmetric) should be processed as missing values.
- **symmetry_minor_diagonal** is a flag, and if set to 1, indicates that the symmetry is along the minor diagonal of the main Table Data. Otherwise, symmetry is along the principal diagonal by default.

7 Dependency and Sparse Transforms

We propose the inclusion of dependency transform and sparse transform in the compression framework. Dependency transform refers to the ability of exploiting dependencies across multiple attributes to improve compression. This can be achieved by transforming one attribute based on the values of another attribute, with the inverse operation being applied during decompression. The simplest such transform is the reorder transform applicable to attributes taking values in a discrete set. This transform sorts the list of one attribute based on the corresponding values of the other attribute, and can theoretically achieve the optimal compression (conditional entropy) under standard assumptions. We demonstrated advantages of this approach for genotype likelihood compression in input document M52159 subclause 4.1.2.4. More specialized transforms include delta coding of start and end positions in a GTF/GFF file based on knowledge of the feature (gene/transcript/exon). In the context of arithmetic coding-based entropy coders (e.g., CABAC), this idea corresponds to using one attribute as a context while compressing the other attribute.

These and other transforms were proposed as part of GPress [2] and incorporated in a standard interface provided in the input documents (M52159 subclause 3.1 and M53381 subclause 6.4.1.3.2). The interface includes flags to denote the use of such a transform, the name of the transform, and the relevant dependency attribute(s). We demonstrated the reduction in compressed size by using these transforms for specific attributes (input document M52159 subclauses 4.1.2.4 and 4.2.4). Note that the use of these transforms needs to be restricted to make sure that decompression is possible, in particular there shouldn't be any cyclic dependencies in the dependency graph. Also, certain transforms are restricted to certain attributes, while others (e.g., reorder transform) are more general. Advantages of compressing attributes dependent on other attributes have also been demonstrated in other research on genomic data compression (e.g., METHCOMP [3]).

In addition to the dependency transform, we also propose including the sparse transform as part of the compression framework, as done in GPress and incorporated in a standard interface provided in the input documents (M52159 subclause 3.1 and M53381 subclause 6.4.1.3.2). The sparse transform simply takes in a stream of values for an attribute along with the default value and stores only the occurrences of the non-default values. The coordinates and values need to be stored, where the nature of the coordinate stream depends on the dimensionality of the data which determines the delta coding applied to the stream. While we exhibit the benefits of this transform on scRNA gene expression data (input document M52159 subclause 4.3.2), this should be available as an option for any attribute via a unified interface.

Some dependency transforms and the sparse transform can produce multiple streams after transformation (e.g., coordinate and value for sparse transform). These can be treated as subsequences which can be encoded using different entropy coding techniques and the framework can be incorporated into the working draft subclause 7.11 (parameter set).

8 References

1. Androulakis, I.P., E. Yang, and R.R. Almon, *Analysis of time-series gene expression data: methods, challenges, and opportunities.* Annu Rev Biomed Eng, 2007. **9**: p. 205-28.
2. Meng, Qingxi, Idoia Ochoa, and Mikel Hernaez, *GPress: a framework for querying General Feature Format (GFF) files and feature expression files in a compressed form*. bioRxiv (2019): 833087.
3. Jianhao Peng, Olgica Milenkovic, and Idoia Ochoa, *METHCOMP: a special purpose compression platform for DNA methylation data, Bioinformatics*, Volume 34, Issue 15, 01 August 2018, Pages 2654–2656, https://doi.org/10.1093/bioinformatics/bty143