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/M53383
April 2020, Alpbach, Austria

Source: ISO/IEC JTC1/SC29/WG11
Status: Input Document
Title: **Philips' Response to CE2 of MPEG-G Part 6**
Author: René van der Vleuten (Royal Philips), Patrick Y.H. Cheung (Royal Philips), Shubham Chandak (Stanford University)

1 Introduction

A unified file format for the coding of genomic annotations was proposed in contribution M52159. Further details of the proposed format are provided in our contribution M53381 to CE1 of MPEG-G Part 6. The complete syntax proposed in CE1 could, in principle, also be used in CE2, but this has the disadvantage that CE2 would have to wait for the CE1 syntax to be agreed and completed. Therefore, to be able to work in parallel to CE1, for the moment it is proposed to use only a minimal sub-set of the CE1 syntax in CE2, as described in the following.

2 Descriptor Stream Syntax and File Format

For CE2 files, for the moment, a minimal sub-set of the proposed CE1 syntax is proposed to be used, as shown in Table 1.

Table 1 – File format structure and encapsulation levels

tdai	
	tdap
tdbl	

The actual attribute data (the "payload") is contained in the table data block (tdbl) structure. For the table data block (tdbl), the value of block_type shall be equal to 0 and all data shall be contained in a single chunk. The table data attribute information (tdai), containing a table data attribute parameter set (tdap) for each attribute, provides a general header and details on the different attributes.

The tdai, tdap, and tdbl structures are explained in detail in our "Response to CE1 of MPEG-G Part 6" contribution M53381. They have been copied in the Annex to the present document for easy reference, where the tdap is shown in Table 2 and Table 3, tdai is shown in Table 4 and Table 5 and tdbl is shown in Table 6.

Since CE2 considers only uncompressed descriptors (with compression being part of CE3), the compressor_ID in tdap is set to 0, meaning "uncompressed" (CE3 can use different values for

the compressor_ID, while re-using the CE2 file format). Furthermore, in tdbl, the block_type shall be equal to 0 and all data shall be placed in a single (large) chunk, such that the data can easily be extracted (e.g. for CE3).

3 Attribute ID definitions

The attribute_ID values (as used in the tdap) are defined separately for each file format supported by MPEG-G Part 6. The definitions for VCF, GTF, and scRNAseq gene expression are given in the following subsections. Definitions for other formats can be added in future.

3.1 VCF (with samples)

Note: for VCF without samples, the auxiliary data (table_data_class = 1) becomes the main Table Data (ignoring the format field)
Note: the comment lines go into the metadata for the table and for the respective INFO and FORMAT attribute_metadata.
Note: attribute_missing_value is "." for the string fields and "0" for integer/bool fields
Note: Names of the chromosomes (string) are stored in the chrom, pos supplementary index to allow random access by chromosome name.
Note: columns in VCF file directly translate to the attributes

3.1.1 table_data_class = 1 (auxiliary Table Data for data attributes mapped to the rows of the main Table Data)

attribute_ID value	attribute_name	attribute_type	attribute_default_value
0	CHROM	Uint16	0
1	POS	Uint32	0
2	ID	String	·
3	REF	String	·
4	ALT	String	·
5	QUAL	String	·
6	FILTER	string	·
7	FORMAT	string	·
8 and beyond	INFO fields	String (except for flag fields, when it's bool)	Depends on INFO field

NOTE: For VCF without samples, FORMAT attribute is not present and INFO fields are 7, 8, …

3.1.2 table_data_class = 2 (auxiliary Table Data for data attributes mapped to the columns of the main Table Data)

attribute_ID value	attribute_name	attribute_type	attribute_default_value
0	SAMPLE	string	0

3.1.3 table_data_class = 0 (main Table Data)

attribute_ID value	attribute_name	attribute_type	attribute_default_value
0 and beyond	FORMAT fields	String (except for flag fields, when it's bool)	Depends on FORMAT field (e.g., can have "0|0" for GT field for diploid organism)

3.1.4 Suggested compression settings (for CE3)

POS: can use delta coding
DS and GL FORMAT fields: can compress dependendent on GT field for improved compression (e.g., using reorder transform).

3.2 GTF

Note: the comment lines go into the metadata for the table.
Note: attribute_missing_value and default values are "." for the string fields, "0" for integer/bool fields and "(0,0)" for start_end field.
Note: columns in GTF file directly translate to the attributes.
Note: Names of the chromosomes (string - SEQNAME) are stored in the chrom, pos supplementary index to allow random access by chromosome name.

3.2.1 table_data_class = 0 (main Table Data)

attribute_ID value	attribute_name	attribute_type
0	SEQNAME	Uint16
1	SOURCE	string
2	FEATURE	String
3	START_END	start_end
4	SCORE	String
5	STRAND	Bool ("+" -> 0, "-" -> 1)
6	FRAME	string
7	ATTRIBUTE	string

3.2.2 Suggested compression settings (for CE3)

START_END: use GPress transform dependent on FEATURE
STRAND: use GPress transform dependent on FEATURE
FRAME: use GPress transform dependent on FEATURE

3.3 scRNAseq gene expression

Note: the comment lines go into the metadata for the table.
Note: attribute_missing_value/default_value is "." for the string fields and "0" for integer fields

3.3.1 table_data_class = 1 (auxiliary Table Data for data attributes mapped to the rows of the main Table Data) (read from features.tsv file)

attribute_ID value	attribute_name	attribute_type
0	GENE_ID	string
1 and beyond	Depends on file (can be set to ATTRIBUTE_1, ATTRIBUTE_2, …)	string

3.3.2 table_data_class = 2 (auxiliary Table Data for data attributes mapped to the columns of the main Table Data) (read from barcodes.tsv file)

attribute_ID value	attribute_name	attribute_type
0	BARCODE	string
1 and beyond	Depends on file (can be set to ATTRIBUTE_1, ATTRIBUTE_2, …)	string

3.3.3 table_data_class = 0 (main Table Data) (read from matrix.mtx file)

attribute_ID value	attribute_name	attribute_type
0	EXPRESSION_VALUE	Uint32

3.3.4 Suggested compression settings (for CE3)

EXPRESSION_VALUE: can use sparse coding

3.4 Other file types

Not implemented currently, but similar principles apply in an obvious manner. E.g., for bed and wig, each column becomes an attribute with the appropriate type.

4 Data for cross-checking

The data for cross-checking is provided on a sharepoint site, which is accessible to the CE2 cross checkers, as well as the CE2 coordinator.

The following MPEG-G Part 6 items have been processed:
08_homo_sapiens_somatic
09_homo_sapiens_structural_variations
10_Selected.integrated_phase1_v3.20101123.snps_indels_svs.genotypes
11_nstd112.GRCh37.variant_call
11_nstd112.GRCh37.variant_region
12_nstd152.GRCh37.variant_call
12_nstd152.GRCh37.variant_region
14_Homo_sapiens.GRCh38.95.chr_patch_hapl_scaff

At the moment the file format implementation is still in progress. Therefore, for each attribute_ID the payload is given in a separate file that contains the binary data (the tdbl payload) and small text files are provided that contain the information that would be stored in the headers of the proposed file format. To decode the attribute data, the decode_attribute.c program is provided. The decoded attribute data, together with the information in the (header) text files, enables the reconstruction of the (equivalent) information of the original input file.

The type of each attribute (used as input in decode_attribute.c) is given in section 3 for the fixed attributes. For the variable attributes a .csv file is provided with the type of each attribute.

5 Annex

This Annex contains the syntax structures, as defined in our contribution to CE1, that apply to CE2. They are copied here for easy reference.

Table 2 – Data Attribute Parameter Set syntax

Syntax	**Key**	**Type**	**Remarks**
table_data_attribute_parameter_set {	*tdap*		
attribute_ID		u(16)	
attribute_name		st(v)	
attribute_metadata		gen_text	
attribute_type		u(8)	
attribute_default_value		st(v)	
attribute_missing_value		st(v)	
compressor_ID		u(8)	
if (transform) {			transform equals 0 for CE2
for (i=0; i<n_dependencies; i++) {			
reserved		u(5)	

dependency_table_data_ID[i]		u(3)	
dependency_attribute_ID[i]		u(16)	
}			
}			
compressor_common_data		compressor_common_data	No data is used for CE2

For CE2, the compressor_ID shall be equal to 0, meaning the data is not compressed.

Table 3 – tdap attribute_type definition

Type	attribute_type u(8)	Number of bytes
Null terminated string	0	variable
Char	1	1
Boolean	2	1
Uint8	3	1
Int8	4	1
Uint16	5	2
Int16	6	2
Uint32	7	4
Int32	8	4
Uint64	9	8
Int64	10	8
Float	11	4
Double	12	8
Start_end (pair of uint32)	13	8

Table 4 – Table Data Attribute Information syntax

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

Table 5 – Table Data Header syntax

Syntax	**Key**	**Type**
table_data_header {		
reserved		u(5)
dataset_group_ID		u(8)
dataset_ID		u(16)
table_ID		u(8)
table_data_ID		u(3)
}		

Table 6 – Table Data Block syntax

Syntax	Key	Type
table_data_block {	*tdbl*	
table_data_header		table_data_header
reserved		u(7)
block_type		u(1)
if (block_type == 0) {		
if (variable_size_chunks || !two_dimensional) {		
chunk_idx_1		u(table_index_size*8)
chunk_idx_2 = 0		
}		
else {		
chunk_idx_1		u(table_index_size*8)
chunk_idx_2		u(table_index_size*8)
}		
for (i=0; i<n_attributes; i++) {		
payload_size[i][chunk_idx_1][chunk_idx_2]		u(32)
payload[i][chunk_idx_1][chunk_idx_2]		u(payload_size[i]*8)
}		
}		
else {		
attribute_ID		u(16)
if (variable_size_chunks || !two_dimensional) {		
for (j=0; j<n_chunks; j++) {		
payload_size[attribute_ID][j]		u(32)
payload[attribute_ID][j]		u(payload_size[i]*8)
}		
}		
else if (column_major_chunk_order) {		
n_chunks_per_col		u(table_index_size*8)
for (k=0; k<n_chunks_per_row; k++) {		
for (j=0; j<n_chunks_per_col; j++) {		
payload_size[attribute_ID][j][k]		u(32)
payload[attribute_ID][j][k]		u(payload_size[i]*8)
}		
}		
}		
else {		
n_chunks_per_row		u(table_index_size*8)
for (j=0; j<n_chunks_per_col; j++) {		
for (k=0; k<n_chunks_per_row; k++) {		
payload_size[attribute_ID][j][k]		u(32)
payload[attribute_ID][j][k]		u(payload_size[i]*8)
}		
}		
}		
}		
}		

For CE2, the block_type, chunk_idx_1 and chunk_idx_2 shall all be equal to 0, and all data shall be placed in a single (large) chunk.