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/M53381
April 2020, Alpbach, AT

Source ISO/IEC JTC1/SC29/WG11
Status Input Document
Title **Philips' Response to CE1 (Phase 1) of MPEG-G Part 6**
Authors Patrick Y.H. Cheung (Royal Philips), Shubham Chandak (Stanford University), René van der Vleuten (Royal Philips)

Introduction

While the existing Parts 1-5 of the ISO/IEC 23092 (MPEG-G) standard deal with the representation of genomic information derived from the primary analysis of high-throughput sequencing (HTS) data – sequencing reads and qualities, and their alignment to a reference genome – that is only the first step in a long series. In particular, the results of primary analysis are usually processed further in order to obtain higher-level information. Such a process of aggregating information deduced from single reads and their alignments to the genome into more complex results is generally known as secondary analysis. In most HTS-based biological studies, the output of secondary analysis is usually represented as different types of annotations associated to one or more genomic intervals on the reference sequences.

Biological studies typically produce genomic annotation data such as mapping statistics, quantitative browser tracks, variants, genome functional annotations, gene expression data and Hi-C contact matrices. These diverse types of downstream genomic data are currently represented in different formats such as VCF, BED, WIG, etc., with loosely defined semantics, leading to issues with interoperability, the need for frequent conversions between formats, difficulty in the visualization of multi-modal data and complicated information exchange. Figure 1 depicts a typical pipeline for the primary and secondary analyses of HTS data, the file formats involved and the scopes of different parts of the ISO/IEC 23092 standard.

Furthermore, the lack of a single format has stifled the work on compression algorithms and has led to the widespread use of general compression algorithms with suboptimum performance. These algorithms do not exploit the fact the annotation data typically comprises of multiple fields (attributes) with different statistical characteristics and instead compress them together. Therefore, while these algorithms support efficient random access with respect to genomic position, they do not allow extraction of specific fields without decompressing all the whole file.

In response to the aforementioned challenges, this document details a unified data format for the efficient representation and compression of diverse genomic annotation data for file storage or data transport. The benefits are manifold: reducing the cost of data storage, improving the speed of random data access and processing, providing support for data security and privacy in selective genomic regions, and creating linkages across different types of genomic annotation and sequencing data. The ultimate goal is to enable the secured and seamless sharing, processing and analysis of multi-modal genomic data in order to reduce the burden of data manipulation and management, so scientists can focus on biological interpretation and discovery.

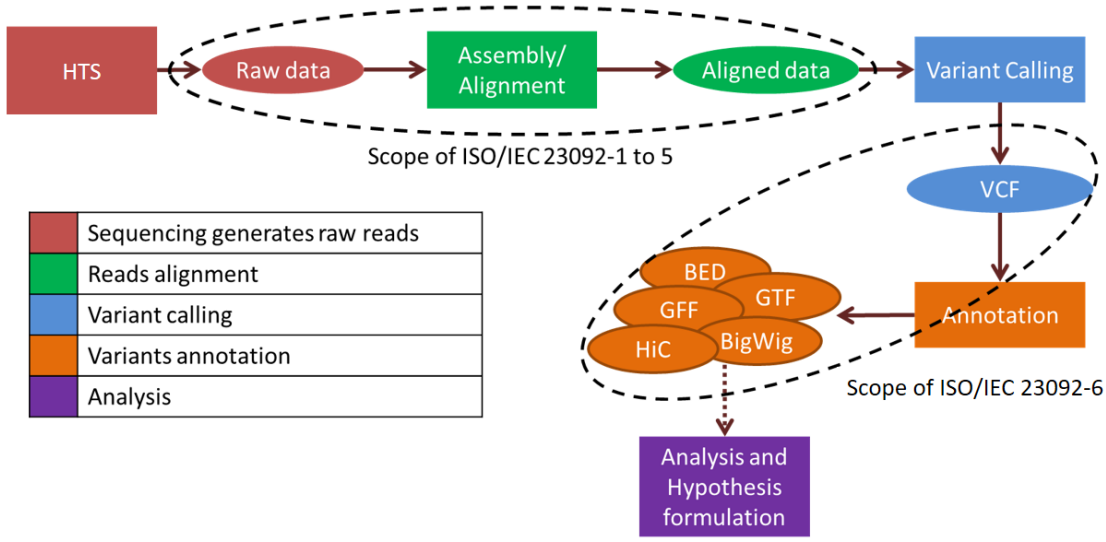

Figure 1 – Typical pipeline for the primary and secondary analyses of HTS data

1 Scope

This document specifies data formats for both transport and storage of genomic annotation data, including but not limited to: genomic variants (VCF), gene expressions, genomic functional annotation (BED, GTF, GFF, GFF3 and GenBank), quantitative browser tracks (Wig, BigWig and BedGraph) and chromosome conformation capture (HiC file), and describes the conversion process from transport to file formats.

2 Normative References

The following documents are referred to in the text in such a way that some or all of their content constitutes requirements of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

ISO/IEC 10646, Information technology — Universal Coded Character Set (UCS)

ISO/IEC/FDIS 23092-1, Information technology — Genomic information representation — Part 1: Transport and storage of genomic information

ISO/IEC/FDIS 23092-2, Information technology — Genomic information representation — Part 2: Coding of genomic information

ISO/IEC/FDIS 23092-3, Information technology — Genomic information representation — Part 3: Metadata and application programming interfaces (APIs)

ISO/IEC/FDIS 23092-4, Information technology — Genomic information representation — Part 4: Reference Software

IETF RFC 3986, Uniform Resource Identifier (URI): Generic Syntax

IETF RFC 7320, URI Design and Ownership

3 Terms and Definitions

For the purposes of this document, the following terms and definitions apply.

ISO and IEC maintain terminological databases for use in standardization at the following addresses:

— ISO Online browsing platform: available at https://www.iso.org/obp

— IEC Electropedia: available at http://www.electropedia.org

3.1 attribute

annotation data field that consists of one or multiple chunks (3.4) on which the same compressor (3.7) is applied. An attribute is defined in Table Data Attribute Parameter Set (3.18) and identified by attribute ID unique within Table Data (3.16).

3.2 attribute contiguity

setting for grouping payloads into Table Data Blocks (3.19) by attribute (3.1) and ordering them by chunk (3.4) according to the choice of chunk order (3.6) if Table Data (3.16) is two dimensional

3.3 box

object-oriented building unit defined by a unique type identifier and length

3.4 chunk

rectangular region corresponding to specific ranges of rows and/or columns defined in Table Data Chunk Structure (3.21)

3.5 chunk contiguity

setting for grouping payloads into Table Data Blocks (3.19) by chunk (3.4) and ordering them by attribute ID (3.1)

3.6 chunk order

method of ordering chunks in two dimensional Table Data (3.16), can be either row-major, with elements arranged from left to right per row and then moving from one row to the next from top to bottom; or column-major, with elements arranged from top to bottom per column and then moving from one column to the next from left to right.

3.7 Compressor

data structure within Dataset Parameter Set (3.12) that contains configuration of transform and compression algorithms to be associated with one or more attributes (3.1) through its unique Compressor ID within the annotation dataset (3.10)

3.8 container box

box (3.3) whose sole purpose is to contain and group a set of related boxes

3.9 data stream

set of packets (3.14) transporting the same data type

3.10 Dataset

container box (3.8) identified by Dataset ID within Dataset Group that contains one or multiple Tables (3.15) for the representation of genomic annotation data if Dataset Type is set to 3. The annotation data is further classified into subtypes that include “VCF”, “GeneExpression”, “Wig”, “BigWig”, “BedGraph”, “BED”, “GTF”, “GFF”, “GFF3”, “GenBank”, “HiC” and other user-defined values.

3.11 Dataset Mapping Table

mandatory box (3.3) under Dataset (3.10) that lists all data streams transporting data related to the dataset identified by Dataset ID

3.12 Dataset Parameter Set

container box (3.8) describing any of the parameter sets associated to the dataset. For annotation datasets, it contains the definition of compressors (3.7) needed for the compression/decompression of attributes in tables.

3.13 file format

set of data structures for the storage of coded information

3.14 packet

transmission unit transporting segments of any of the data structures defined in this document

3.15 Table

container box (3.8) in annotation dataset (3.10) identified by Table ID and comprising tabulated annotation data that includes a main Table Data (3.16) and optionally one or multiple auxiliary Table Data

3.16 Table Data

container box (3.8) in Table (3.15) identified by Table Data ID and grouping attributes into classes: 0 – main; 1/2 – auxiliary row/column data attributes; 3/4 – auxiliary row/column linkage attributes; 4-7 – user-defined auxiliary attributes

3.17 Table Data Attribute Information

container box (3.8) in Table Data (3.16) that comprises a collection of attribute definitions encapsulated in Table Data Attribute Parameter Sets (3.18)

3.18 Table Data Attribute Parameter Set

box (3.3) in Table Data Attribute Information (3.17) that contains the basic information of an attribute (3.1) and its associated compressor (3.7)

3.19 Table Data Block

box (3.3) in Table Data (3.16) that groups and organizes the compressed payloads. There are two types of Table Data Block: Type 0 for chunk contiguity (3.5), where a block contains payloads of the same chunk ordered by their attribute IDs; and Type 1 for attribute contiguity (3.2), where a block contains payloads of the same attribute ordered by their chunk indices.

3.20 Table Data Byte Offset

box (3.3) in Table Data Master Index (3.23) that comprises the byte-offset pointers to the Table Data Blocks (3.19) and their individual payloads

3.21 Table Data Chunk Structure

box (3.3) in Table Data Master Index (3.23) that contains information on how Table Data (3.16) should be divided into rectangular chunks (3.4). The chunk size, in term of number of rows/columns, can be uniform, in which case only the size per dimension needs to be specified, or variable, in which case the ranges of row and/or column indices need to be specified for individual chunks.

3.22 Table Data Header

mandatory data structure in the transport format (3.26) for the four boxes (3.3) – Table Data Attribute Information (3.17), Table Data Master Index (3.23), Table Data Supplementary Indices

(3.24) and Table Data Block (3.19) – in Table Data (3.16). It contains the IDs of the upper-level containers that are required for the assembly of the Table Data structures after transport, but is excluded from the file format (3.13).

3.23 Table Data Master Index

container box (3.8) in Table Data (3.16) that carries indexing information consisting of one or multiple (if attribute-dependent) Table Data Chunk Structure (3.21) boxes, and a Table Data Byte Offset (3.20) box. It enables the mapping between row and/or column indices of Table Data and specific chunks (3.4) of an attribute (3.1).

3.24 Table Data Supplementary Indices

optional container box (3.8) in Table Data (3.16) that carries additional attribute-specific indexing data encapsulated in Table Data Supplementary Index Data (3.25) for enabling query search based on criteria such as genomic region, gene symbol or any other attributes

3.25 Table Data Supplementary Index Data

box (3.3) in Table Data Supplementary Indices (3.24) that contains information and data of a supplementary index

3.26 transport format

set of data structures for the transport of coded information

3.27 variable

parameter either inferred from syntax fields or locally defined in a process description

4 Mathematical Operators

NOTE The mathematical operators used in this document are similar to those used in the C programming language. However, integer division with truncation and rounding are specifically defined. The bitwise operators are defined assuming two's-complement representation of integers. Numbering and counting loops generally begin from 0.

4.1 Arithmetic operators

+	addition
−	subtraction (as a binary operator) or negation (as a unary operator)
++	increment
*	multiplication

/ integer division with truncation of the result toward 0 (for example, 7/4 and −7/−4 are truncated to 1 and −7/4 and 7/−4 are truncated to −1)

4.2 Logical operators

|| logical OR

&& logical AND

! logical NOT

4.3 Relational operators

> greater than

≥ greater than or equal to

< less than

≤ less than or equal to

== equal to

!= not equal to

4.4 Bitwise operators

& AND

| OR

>> shift right with sign extension

<< shift left with 0 fill

4.5 Assignment

= assignment operator

4.6 Unary operators

sizeof(N) size in bytes of N, where N is either a data structure or a data type

5 Overview of Coded Genomic Annotation Data

5.1 Basic components of an annotation table – attributes and chunks

An annotation table is divided into *attributes*, each of which contains data with similar statistical characteristics and is therefore independently compressed to improve the compression ratio. Usually, an attribute represents a column, but if a column contains multiple data fields, such as the INFO column of a VCF file, then an attribute can represent an individual data field within a column. Figure 2 shows two table examples, with x_{ij}, y_{ij} and z_{ij} denoting three data fields with different statistical characteristics, where (i, j) are the row and column indices. The table in (a) can be divided into three one-dimensional (1-d) attributes corresponding to data fields x, y and z. Whereas the table in (b) can be divided into three two-dimensional (2-d) attributes corresponding to the same data fields.

Column A	Column B	Column C
x_{00}	y_{01}	z_{02}
x_{10}	y_{11}	z_{12}
⋮	⋮	⋮
x_{n0}	y_{n1}	z_{n2}

(a)

Column A	Column B	Column C
$x_{00}; y_{00}; z_{00}$	$x_{01}; y_{01}; z_{01}$	$x_{02}; y_{02}; z_{02}$
$x_{10}; y_{10}; z_{10}$	$x_{11}; y_{11}; z_{11}$	$x_{12}; y_{12}; z_{12}$
⋮	⋮	⋮
$x_{n0}; y_{n0}; z_{n0}$	$x_{n1}; y_{n1}; z_{n1}$	$x_{n2}; y_{n2}; z_{n2}$

(b)

Figure 2 – Examples of (a) one-dimensional and (b) two-dimensional table attributes

To support selective data access, the data of each attribute can be further divided into rectangular regions known as *chunks*, each of which with independent compressor configurations for optimum performance. With a master index that provides the mapping between table indices and chunks, data from selected regions of the table can be accessed by looking up and decompressing only the chunks that overlap with the region without the need to decompress the whole file.

It is flexible for user to define the layout of the chunks using a *chunk structure*. The size of each chunk can be uniform or variable across the table. Usually the same chunk structure is applied to all attributes for ease of indexing. However, in cases where the attributes have widely different characteristics, *attribute-dependent chunk structures* can be applied. In general, a larger chunk size improves the compression ratio but reduces the speed of selective access. Figure 3 and Figure 4 show the example chunk structures of respectively uniform- and variable-size chunks.

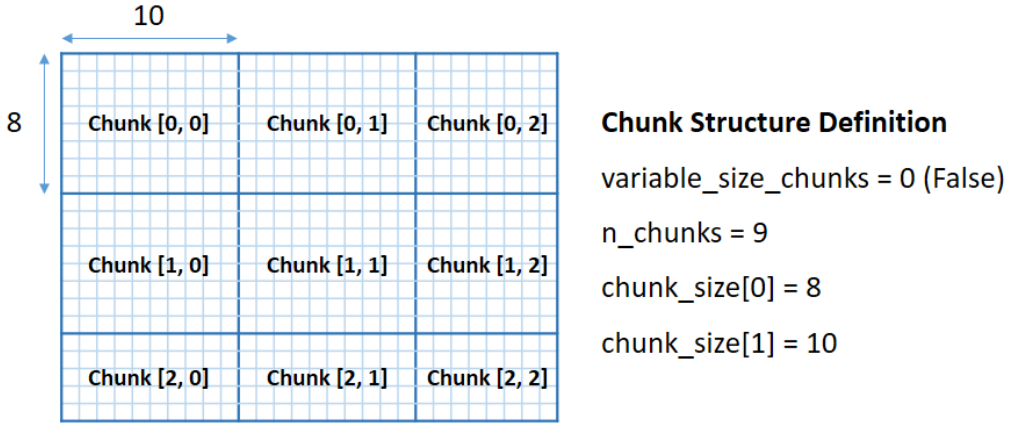

Figure 3 – Example chunk structure of uniform-size chunks

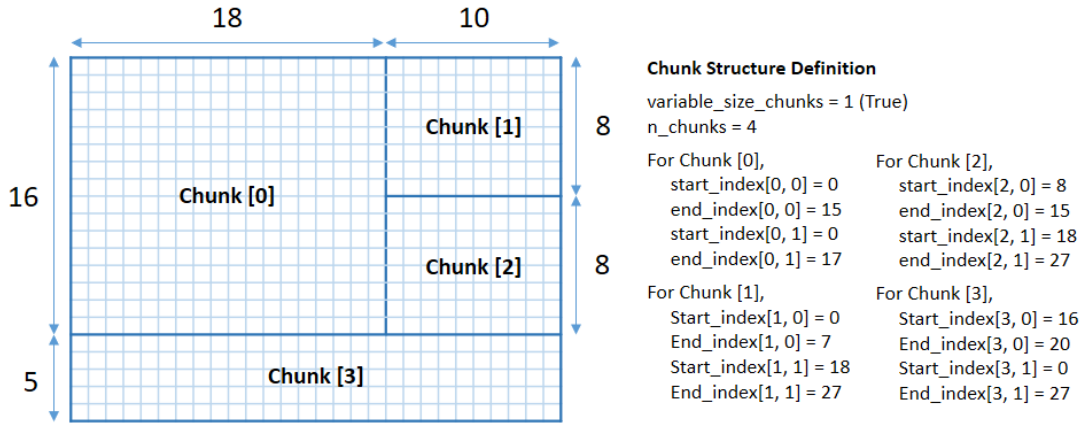

Figure 4 – Example chunk structure of variable-size chunks

5.2 Key containers for annotation data

5.2.1 Dataset

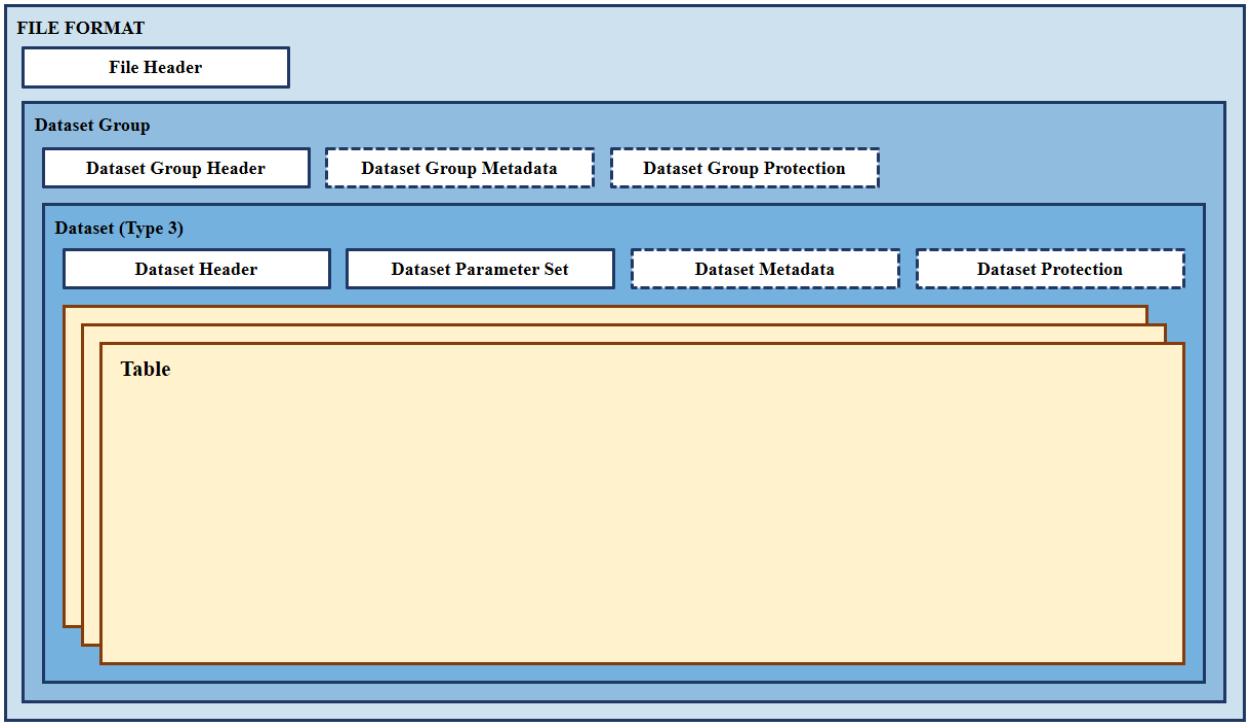

Figure 5 – Top-level container hierarchy for annotation datasets (Type 3)

In this format, the top-level container boxes: File, Dataset Group, and Dataset from Part 1 are retained, with extensions to: (1) Dataset to include a new Dataset Type of value 3 for the representation of genomic annotation data, and (2) Dataset Parameter Set to store compressor configurations for use on selected attributes. A dataset can contain multiple (< 128) tables, which may correspond to data at different resolutions, sampling time points, etc. Figure 5 shows the top-level container hierarchy for annotation datasets.

5.2.2 Table

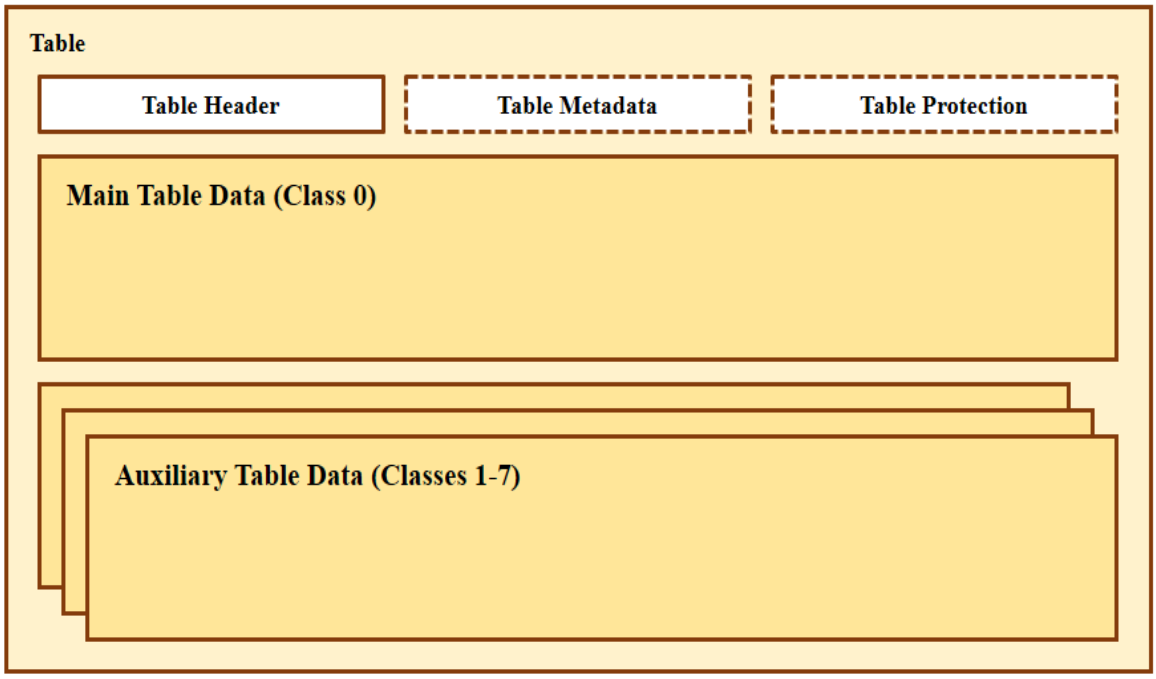

Figure 6 – Overview of Table container structure

Table, identified by Table ID, is the main container box that is specific to annotation data and consists of the following components as shown in Figure 6:

— Table Header, a mandatory box describing the content of the Table

— Table Metadata, an optional box containing metadata associated with the Table, which includes some basic information and metadata that supports functionalities such as data traceability, reproducibility and linkages with other datasets or tables

— Table Protection, an optional box containing protection information associated with the Table to support confidentiality (encryption), integrity verification (digital signature) and access control policy enforcement

— Main Table Data (Class 0), a mandatory box containing the core attributes of the Table

— Auxiliary Table Data (Classes 1-7), one or multiple (< 8) optional boxes containing the auxiliary attributes that supplement the Main Table Data. The classes of auxiliary Table Data include: 1/2 for auxiliary row/column data attributes, 3/4 for auxiliary row/column linkage attributes, and 4-7 for user-defined auxiliary attributes

5.2.3 Table Data

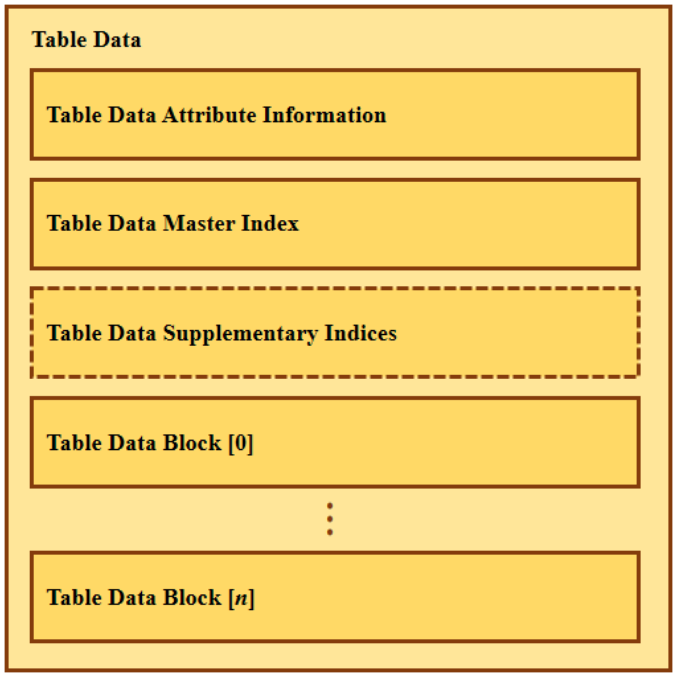

Figure 7 – Overview of Table Data container structure

Table Data, identified by Table Data ID, is a container box in Table that groups attributes into classes by their roles, and consists of the following components as shown in Figure 7:

— Table Data Attribute Information, a mandatory box containing a collection of attribute definitions

— Table Data Master Index, a mandatory box containing indexing information that enables the mapping between row and/or column indices of Table Data and specific chunks of an attribute

— Table Data Supplementary Indices, an optional box containing additional attribute-specific indexing data that enables query search based on criteria such as genomic region, gene symbol or any other attributes

— Table Data Blocks, mandatory boxes that groups and organizes the compressed payloads. There are two types of Table Data Block: Type 0 for chunk contiguity, where a block contains payloads of the same chunk ordered by their attribute IDs; and Type 1 for attribute contiguity, where a block contains payloads of the same attribute ordered by their chunk indices.

Compound query that consists of a logical combination of attribute conditions can be realized by (1) looking up the row and/or column indices satisfying each attribute condition independently, (2) identifying the subset of indices satisfying the logics in the compound query, (3) mapping the subset of indices to specific chunks of an attribute, and (4) looking up the locations of the payloads of the matching chunks.

5.2.4 Table Data Block

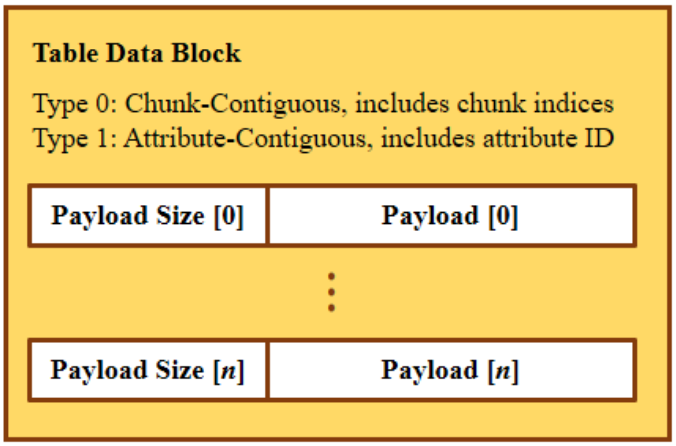

Figure 8 – Overview of Table Data Block container structure

Table Data Block is a container box in Table Data that groups and organizes the compressed payloads. For chunk-contiguous (Type 0) blocks, index (1-d) or indices (2-d) of the chunk to which the payloads belong are included. Whereas for attribute-contiguous (Type 1) blocks, the ID of the attribute to which the payloads belong is included. They are then followed by a concatenation of the size and data of the compressed payloads. The order of the payloads should be the same as the order of the attributes in Table Data Attribute Information for chunk-contiguous blocks, or the order of the chunks for attribute-contiguous blocks. For 2-d Table Data, either row-major or column-major chunk ordering can be applied.

6 Data Format

6.1 Format structure

6.1.1 General

Table 1 presents the overall data structures and hierarchical encapsulation levels for the representation of genomic annotation data built on and compatible with ISO/IEC 23092-1 (Part 1). Data structures that remain the same are in gray text, with the reference to the corresponding clause in Part 1 included. Boxes that may occur at the top-level are shown in the left-most column; indentation is used to show possible containment. Not all boxes need to be used in all files; the mandatory boxes are marked with an asterisk (*) in the Mandatory column: such column refers to the relevant scope (File and/or Transport). Optional boxes are represented with dashed borders in Figure 9 and Figure 10. Mandatory boxes are represented with solid borders. When no entry is present in the Scope column, scope is both File and Transport. If the box key is represented in italic format in Table 1, the relevant box is represented with neither Key nor Length, but only Value in the gen_info format, as specified in subclause 6.3.1, for all boxes but offset, as specified in subclause 6.6.4.1 of Part 1 for the offset box.

Table 1 – Format structure and encapsulation levels

Box key (with hierarchical level)						Sub-clause	Scope	Mandatory
0	1	2	3	4	5			
flhd						Part 1: 6.5.1	File	*

dgcn						Part 1: 6.5.2	File	*
	dghd					Part 1: 6.5.2.2		*
	dgmd					Part 1: 6.5.2.6		
	dgpr					Part 1: 6.5.2.7		
	dmtl					Part 1: 6.7.3	Transport	*
	dtcn					6.4.1	File	*
		dthd				6.4.1.2		*
		pars				6.4.1.3		*
		dtmd				Part 1: 6.5.3.3		
		dtpr				Part 1: 6.5.3.4		
		dmtb				6.6.3	Transport	*
		tbcn				6.4.2	File	*
			tbhd			6.4.2.2		*
			tbmd			6.4.2.3		
			tbpr			6.4.2.4		
			tdcn			6.4.3	File	*
				tdai		6.4.4		*
					table_data_header	6.6.4	Transport	*
					tdap	6.4.4.2		*
				tdmi		6.4.5		*
					table_data_header	6.6.4	Transport	*
					tdcs	6.4.5.2		*
					tdbo	6.5.2	File	*
				tdsi		6.4.6		
					table_data_header	6.6.4	Transport	
					tdsd	6.4.6.2		
				tdbl		6.4.7		*
					table_data_header	6.6.4	Transport	*
	offset	*offs*				Part 1: 6.6.4	File	
packet						Part 1: 6.7.5	Transport	*
	packet_ header					Part 1: 6.7.5.2	Transport	*

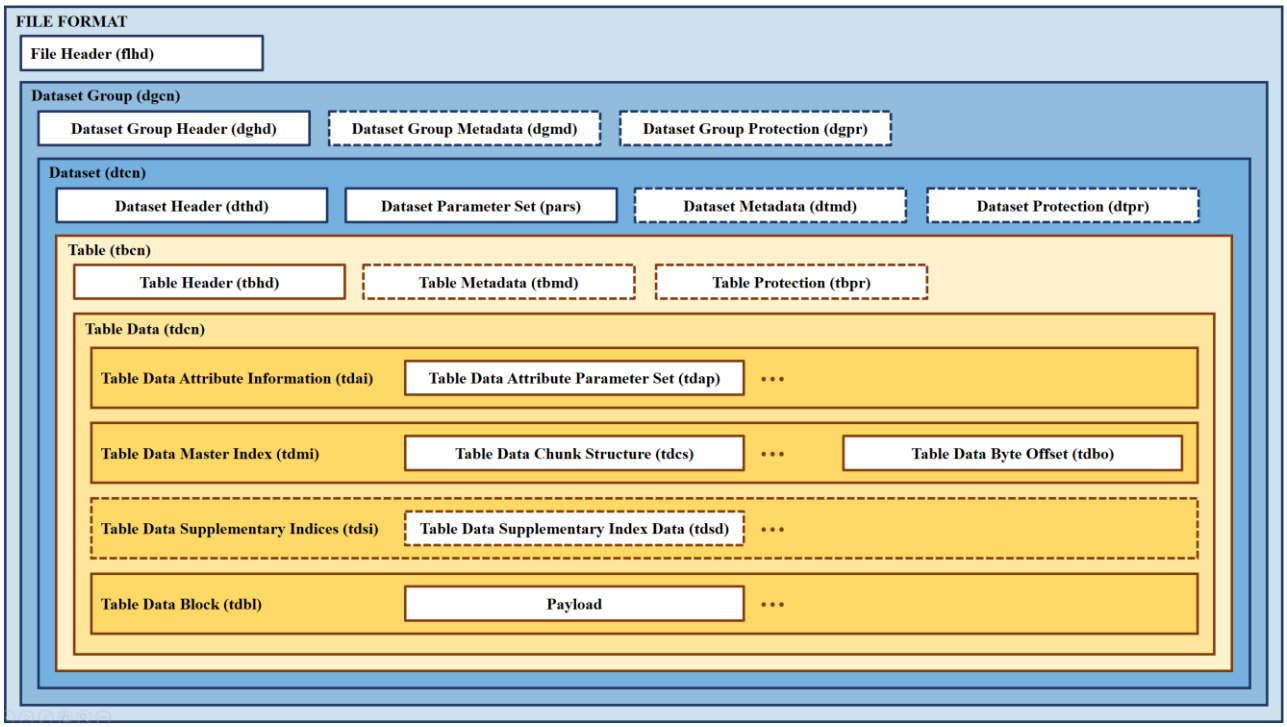

Figure 9 – Data structures hierarchy for storage

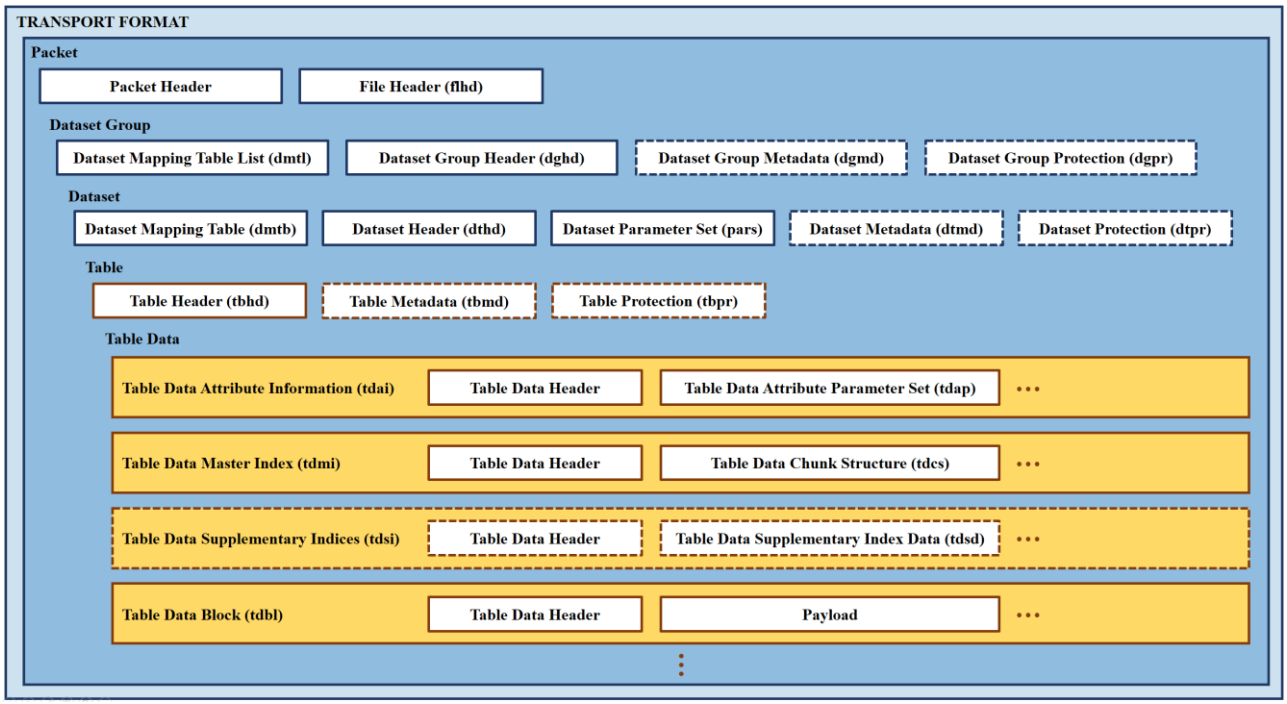

Figure 10 – Data structures hierarchy for transport

Individual data fields of the data types defined in subclauses 6.2.3 and 6.3 – f(n), u(n), st(v), c(n), gen_info and gen_text – are considered an integral part of their containers with the same scope and mandatory statuses, and are therefore not explicitly depicted in Figure 9 and Figure 10.

In transport format, any box represented in Figure 10 shall be encapsulated in one or more packets, as specified in subclause 6.7.5 of Part 1. Dataset Group, Dataset, Table and Table Data are

represented in Figure 10 for clarity, but the corresponding container boxes (dgcn, dtcn, tbcn and tdcn) do not exist in transport format.

6.1.2 Box order

In order to improve interoperability, the following rules shall be followed for the order of boxes:

In file format

1) The container boxes (Dataset Group, Dataset, Table and Table Data) shall be ordered according to the hierarchy specified in Table 1.

2) The box order inside the containers dgcn, dtcn, tbcn, and tdcn are specified in Table 9 of Part 1, Table 4, Table 8 and Table 12, respectively.

3) The file header box ‘flhd’ shall occur before any variable-length box.

4) When present, the offset box ‘offs’, as specified in subclause 6.6.4 of Part 1, enables an indirect addressing of boxes, which, while logically respecting the ordering specified in this subclause, may be physically located in a different position in the file.

5) The contiguity of child boxes inside the containers dgcn, dtcn, tbcn, and tdcn shall not be broken by any box external to the container box, apart from the offset box, as specified in subclause 6.6.4 of Part 1.

In transport format

1) The box order is not specified, but the dataset_mapping_table_list and dataset_mapping_table boxes shall be decoded first, and then all other boxes according to the hierarchy specified in Table 1.

2) It is strongly recommended to transmit the dataset_mapping_table_list, dataset_mapping_table and file_header boxes first.

3) It is recommended to transmit the boxes in hierarchical order, as specified in Table 1.

4) It is recommended, within Table Data, to transmit Table Data Attribute Information and Table Data Master Index first, before transmitting Table Data Blocks and the optional Table Data Supplementary Indices.

6.2 Syntax and semantics

6.2.1 Method of specifying syntax in tabular form

Table 2 lists the constructs that are used to express the conditions when data elements are present.

NOTE This syntax uses the convention that a variable or expression evaluating to a non-zero value is equivalent to a condition that is true.

Table 2 – Constructs used to express the conditions when data elements are present

Construct	Description

while (condition) {
data_element
. . .
}
	If the condition is true, then the group of data elements occurs next in the data stream. This repeats until the condition is not true.

do {
data_element
. . . }
while (condition)
	The data element always occurs at least once. The data element is repeated until the condition is not true.

if (condition) {
data_element
. . .
}
	If the condition is true, then the first group of data elements occurs next in the data stream.

else {
data_element
. . .
}
	If the condition is not true, then the second group of data elements occurs next in the data stream.

for (i=0; i<n; i++) {
data_element
. . .
}
	The group of data elements occurs n times. Conditional constructs within the group of data elements may depend on the value of the loop control variable i, which is equal to zero for the first occurrence, incremented to 1 for the second occurrence, and so forth.

As noted, the group of data elements may contain nested conditional constructs. For compactness, the {} are omitted when only one data element follows. Collections of data elements are represented as listed in Table 3.

Table 3 – Syntax used to represent collections of data elements

data_element[]	data_element[] is an array of data. The number of data elements is indicated by the semantics.
data_element[n]	data_element[n] is the (n+1) th element of an array of data.
data_element[m][n]	data_element[m][n] is the (m+1) th, (n+1) th element of a two-dimensional array of data.
data_element[l][m][n]	data_element[l][m][n] is the (l+1) th, (m+1) th, (n+1) th element of a three-dimensional array of data.

6.2.2 Bit ordering

The bit order of syntax fields in the syntax tables is specified to start with the most significant bit (MSB) and proceed to the least significant bit (LSB).

6.2.3 Specification of syntax functions

read_bits(n) reads the next n bits from the bitstream and advances the bitstream pointer by n bit positions. When n is equal to 0, read_bits(n) is specified to return a value equal to 0 and to not advance the bitstream pointer.

The following data types specify the parsing process of each syntax element:

— f(n): fixed-pattern bit string using n bits written (from left to right) with the left bit first. The parsing process for this data type is specified by the return value of the function read_bits(n).

— u(n): unsigned integer using n bits. When n is "v" in the syntax table, the number of bits varies in a manner dependent on the value of other syntax elements. The parsing process for this data type is specified by the return value of the function read_bits(n) interpreted as a binary representation of an unsigned integer with most significant bit written first.

— st(v): null-terminated string encoded as universal coded character set (UCS) transmission format-8 (UTF-8) characters as specified in ISO/IEC 10646. The parsing process is specified as follows: st(v) reads and returns a series of bytes from the bitstream, beginning at the current position and continuing up to but not including the next byte that is equal to 0x00, and advances the bitstream pointer by (stringLength + 1) * 8 bit positions, where stringLength is equal to the number of bytes returned. The maximum value of stringLength is 16384.

— c(n): sequence of n ASCII characters as specified in ISO/IEC 10646.

6.3 Syntax for representation

6.3.1 General Information (gen_info) data structure

KLV (Key Length Value) format is used for all the data structures listed in Table 1, except table_data_header, offset, packet and packet_header.

The KLV syntax is defined as follows:

```
struct gen_info
{
        c(4)        Key;
        u(64)       Length;
        u(8)        Value[];
}
```

The Length field specifies the number of bytes composing the entire gen_info structure, including all three fields Key, Length and Value.

The table_data_header, offset, packet and packet_header data structures have no Key and no Length, but only Value.

All syntax tables specified in subclauses 6.4, 6.5 and 6.6, for boxes of type gen_info, represent the internal syntax of the Value[] array field only. In the scope of this document the Value[] array is referred as just Value.

6.3.2 General Text (gen_text) data structure

The data type gen_text is for the representation of general text in the syntax tables specified in this document, with the option of having the text compressed or not. Its syntax is defined as follows:

```
struct gen_text
{
        u(23)       Length;
        u(1)        Compression_Flag;
        u(8)        Data[];
}
```

The Length field specifies the number of bytes composing the entire gen_text structure. If Compression_Flag == 1, Data[] consists of data bytes compressed with the default compressor (Compressor ID == 1) defined in dataset_parameter_set. Otherwise, Data[] consists of uncompressed characters.

6.4 Data structures common to file format and transport format

6.4.1 Dataset

6.4.1.1 General

The Dataset structure defined in subclause 6.5.3 of Part 1 is extended as described in this section to support a new dataset type, with dataset_type equal to 3, for the representation of genomic annotations. The new dataset type is a collection of data tables of the same nature, where different tables can store the data at multiple resolutions, among other possible applications. The extended Dataset structure allows various genomic annotation data and high-throughput sequencing (HTS) data to be stored in a unified file format, and ensures backward compatibility with ISO/IEC 23092 Series (Second Edition) and interoperability with existing MPEG-G components.

The relevant container box (dtcn in Table 1) is mandatory in file format, forbidden in transport format.

Child boxes may be present or not, according to the column "Mandatory" in Table 1. Child boxes marked with suffix "[]" after their name in the Syntax column of Table 4 may be present in multiple instances.

Table 4 – Dataset syntax

Syntax	Key	Type	Remarks
dataset {	*dtcn*		
dataset_header	dthd	gen_info	As specified in 6.4.1.2
DT_metadata	dtmd	gen_info	As specified in Part 1: 6.5.3.3
DT_protection	dtpr	gen_info	As specified in Part 1: 6.5.3.4
dataset_parameter_set[]	pars	gen_info	As specified in 6.4.1.3
if (dataset_type < 3) {			
/* Same syntax as in Part 1: Table 19 */			As specified in Part 1: 6.5.3.1
}			
else {			
For (i=0; i<n_tables; i++)			
table[i]	tbcn	gen_info	As specified in 6.4.2
}			
}			

6.4.1.2 Dataset Header

6.4.1.2.1 General

This is a mandatory box describing the content of a Dataset. This data structure is extended based on Part 1: 6.5.3.2 to handle the representation of genomic annotations, with a new dataset_type value of 3.

6.4.1.2.2 Syntax

Table 5 – Dataset Header syntax

Syntax	Key	Type	Remarks
dataset_header {	*dthd*		
dataset_group_ID		u(8)	
dataset_ID		u(16)	
version		c(4)	
if (version[0..1] == '19') {			
/* Same syntax as in Part 1: Table 20 */			As specified in Part 1: 6.5.3.2
}			
else {			
dataset_type		u(4)	
if (dataset_type < 3) {			
/* Same syntax as in Part 1: Table 20, except with the data_type row removed */			As specified in Part 1: 6.5.3.1
}			
else if (dataset_type == 3) {			
reserved		u(4)	
dataset_subtype		st(v)	
dataset_name		st(v)	
dataset_version		st(v)	
byte_offset_size_flag		u(1)	
n_tables		u(7)	
for (i=0; i<n_tables; i++) {			The order of table_ID and table_info should be the same as table in Dataset as specified in 6.4.1.1.
table_ID[i]		u(8)	
table_info[i]		gen_text	
}			
}			
}			
}			

6.4.1.2.3 Semantics

dataset_group_ID is the identifier of dataset group containing the dataset including this Dataset Header.

dataset_ID is the identifier of the dataset. Its value shall be one of the dataset_IDs listed in the Dataset Group Header.

version is the combination of version number, amendment number and corrigendum number of ISO/IEC 23092-2 to which the Value field of the dataset, as specified in subclause 0, complies, and is specified as follows:

— first two bytes: version number, as the last two digits of the year of release of the major brand

— third byte: amendment number, as integer counter from 0 to 9, 0 if no amendment yet

— fourth byte: corrigendum number, as integer counter from 0 to 9, 0 if no corrigendum yet

dataset_type specifies the type of data encoded in the dataset. The possible values are: 0 = non-aligned content; 1 = aligned content; 2 = reference; 3 = annotation.

dataset_subtype specifies the type of genomic annotation data encoded in the dataset. The possible values include: “VCF”, “GeneExpression”, “Wig”, “BigWig”, “BedGraph”, “BED”, “GTF”, “GFF”, “GFF3”, “GenBank”, “HiC” and other user-defined values. Each dataset_subtype is associated with a set of attribute parameter definitions specific to the corresponding genomic annotation file type.

dataset_name is the name of the dataset, which could be the name of the original annotation file.

dataset_version is the version of the dataset for keeping track of updates to the dataset.

byte_offset_size_flag: if equal to 0, the variable byteOffsetSize used in Table Data Byte Offset, as specified in subclause 6.5.2, and representing the number of bits used to encode the fields named data_block_byte_offset and payload_byte_offset, is equal to 32; if set to 1, the variable byteOffsetSize is equal to 64.

n_tables specifies the number of tables in the dataset. Multiple tables can be used to store the data at different resolutions, among other possible applications.

table_ID is the identifier of a table unique within the dataset.

table_info stores the general information, e.g. data resolution, on a table.

6.4.1.3 Dataset Parameter Set

6.4.1.3.1 General

Dataset Parameter Set is a mandatory box describing any of the parameter sets associated to the dataset as specified in subclause 6.5.3.5 of Part 1. While its overall syntax remains the same, its embedded encoding_parameters() structure is extended, as described in Tables Table 5 and Table 6, to support the definition of compressors needed for the decompression of attributes in the tables of annotation datasets.

6.4.1.3.2 Syntax

Table 6 – Encoding parameters syntax

Syntax	Type	Remarks
encoding_parameters() {		
dataset_type	u(4)	
if (dataset_type < 3) {		
/* Same syntax as in Part 2: Table 7 */		
}		
else {		
reserved	u(4)	
n_compressors	u(8)	
for (i=0; i < n_compressors; i++)		
compressor[i]	compressor	
}		
}		

Table 7 – Compressor syntax

Syntax	Type	Remarks
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)	
}		
}		

6.4.1.3.3 Semantics

dataset_type specifies the type of data in the dataset for which the encoding parameters are defined. The possible values are: 0 = non-aligned content; 1 = aligned content; 2 = reference; 3 = annotation.

n_compressors specifies the number of compressors, i.e. configurations of transform and compression algorithms, defined for the annotation dataset.

compressor_ID is the unique identifier of the compressor within the dataset, with the values 0 and 1 reserved respectively for no compression and default compressor. It is used in Table Data Attribute Parameter Set as specified in subclause 6.4.4.2 to associate the corresponding configuration of transform and compression algorithms with an attribute.

transform is a flag, and if set to 1, indicates that the compressor involves data transform before compression. Otherwise, no data transform is involved.

transform_algorithm_ID is the identifier of the transform algorithm being applied, optionally followed by a comma and then a URI that points to the codes of the transform algorithm. The URI shall be compliant with IETF RFC 3986 and IETF RFC 7320. If the ID is known and the codes are already installed, an MPEG-G compliant software can directly perform the transform/inverse-transform operation. If the ID is unknown and a URI is available, then the software should prompt the user to download and install the codes, and register the ID and a pointer to the executables for future use. If the ID is unknown and there is no URI, then the software should inform the user that the algorithm is not available.

n_dependencies specifies the number of dependency attributes for the transform.

n_compression_algorithms specifies the number of compression algorithms applied on an attribute in sequential order.

compression_algorithm_ID[i] is the identifier of the i-th compression algorithm being applied, optionally followed by a comma and then a URI that points to the codes of the compression algorithm. The URI shall be compliant with IETF RFC 3986 and IETF RFC 7320. If the ID is known and the codes are already installed, an MPEG-G compliant software can directly perform

the transform/inverse-transform operation. If the ID is unknown and a URI is available, then the software should prompt the user to download and install the codes, and register the ID and a pointer to the executables for future use. If the ID is unknown and there is no URI, then the software should inform the user that the algorithm is not available.

compression_algorithm_pars[i] is a string of parameters in a predefined format required by the i-th compression algorithm.

The following is an example of compressor configurations for the compression of a sparse matrix, such as genotype values or gene expressions:

n_compression_algorithms = 3

compression_algorithm_ID = {“sparse”, “gzip”, “7-zip”}

compression_algorithm_pars = {“out_streams: coordinates, values”,
“in_streams: coordinates”,
“in_streams: values”}

The compression process is as follows:

(1) Apply the first “sparse” algorithm on the sparse matrix to generate two output streams: “coordinates” and “values” that correspond to the coordinates and values of the cells with non-zero entries,

(2) Apply the second “gzip” algorithm on the “coordinates” stream to generate a stream that consists of the size of type u(32) and payload of the compressed coordinates,

(3) Apply the third “7-zip” algorithm on the “values” stream to generate a stream that consists of the size of type u(32) and payload of the compressed values, and

(4) Output the size and payload of the compressed coordinates, followed by the size and payload of the compressed values.

The process can be reversed for decompression:

(1) Extract the compressed coordinates and values payloads with their known sizes,

(2) Decompress the payloads respectively using the “gzip” and “7-zip” algorithms, and

(3) Reconstruct the original sparse matrix from the decompressed coordinates and values.

6.4.2 Table

6.4.2.1 General

Table is the main container box of tabulated annotation data and always includes the main Table Data, which contains core attribute data, which can be 1-d (two_dimensional_main == 0) or 2-d (two_dimensional_main == 1). One or multiple auxiliary Table Data can be included to supplement the main Table Data with attributes associated with the rows/columns for additional data or linkage information, or for other purposes as defined by users.

Table 8 – Table syntax

Syntax	Key	Type	Remarks
table {	*tbcn*		
table_header	tbhd	gen_info	As specified in 6.4.2.2
table_metadata	tbmd	gen_info	As specified in 6.4.2.3
table_protection	tbpr	gen_info	As specified in 6.4.2.4
table_data_main	tdcn	gen_info	As specified in 6.4.3
for (i=0; i<n_aux_data; i++)			
aux_table_data[i]	tdcn	gen_info	As specified in 6.4.3
}			

6.4.2.2 Table Header

6.4.2.2.1 General

This is a mandatory box describing the content of a Table.

6.4.2.2.2 Syntax

Table 9 – Table Header syntax

Syntax	Key	Type	Remarks
table_header {	*tbhd*		
dataset_group_ID		u(8)	
dataset_ID		u(16)	
table_ID		u(8)	
table_info		gen_text	
n_summary_statistics		u(8)	
for (i=0; i<n_summary_statistics; i++) {			
summary_statistic_key[i]		st(v)	
summary_statistic_value[i]		st(v)	
}			
reserved		u(5)	
two_dimensional_main		u(1)	
symmetry_mode		u(3)	
symmetry_minor_diagonal		u(1)	
table_index_size		u(3)	
n_aux_data		u(3)	
for (i=0; i<n_aux_data; i++) {			
aux_data_name[i]		st(v)	
aux_data_metadata[i]		gen_text	
}			
}			

6.4.2.2.3 Semantics

dataset_group_ID is the identifier of dataset group containing the dataset including this Dataset Header.

dataset_ID is the identifier of the dataset. Its value shall be one of the dataset_IDs listed in the Dataset Group Header.

table_ID is the unique identifier of the table within the dataset. Its value shall be one of the table_IDs listed in the Dataset Header.

table_info is the textual information about the table.

n_summary_statistics specifies the number of summary statistics for the table.

(summary_statistic_key[i], summary_statistic_value[i]) is the key-value pair of the i^{th} summary statistic of the table.

two_dimensional_main is a flag, and if set to 1, indicates that all attributes in the main Table Data are 2-d. Otherwise, all attributes in the main Table Data are 1-d.

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.

table_index_size specifies the number of bytes required for representing the row/column index of the table. It determines the size of the fields n_chunks, chunk_size, start_index and end_index in Table Data Chunk Structure, as specified in subclause 6.4.5.2.

n_aux_data specifies the number of auxiliary Table Data structures in the Table.

aux_data_name[i] is the name of the i^{th} auxiliary Table Data.

aux_data_metadata[i] is the metadata associated with the i^{th} auxiliary Table Data.

6.4.2.3 Table Metadata

6.4.2.3.1 General

Table Metadata is an optional box containing metadata associated with a Table. In addition to some basic information about the table, it can also contain metadata that supports functionalities such as data traceability, reproducibility and linkages with other datasets or tables.

6.4.2.3.2 Syntax

Table 10 – Table Metadata syntax

Syntax	Key	Type	Remarks
table_metadata {	*tbmd*		
dataset_group_ID		u(8)	
dataset_ID		u(16)	
table_ID		u(8)	
TB_metadata_value()			
}			

6.4.2.3.3 Semantics

TB_metadata_value() contains compressed table metadata. The decoding process is specified in Clause 9 of Part 3. The output of the decoding process is an XML document with an element Table as root. The XML schema for table metadata will be provided in a separate document. A table metadata element overwrites the corresponding element whose values differ from the one indicated at the dataset level (i.e., the new value in the table is a specialization of the value at the dataset level). Table 2 of Part 3 defines the process to obtain the dataset metadata with inherited elements. The same approach for metadata protection and mechanism for extensions of the metadata as specified in subclauses 6.5 and 6.6 are applicable to table metadata.

6.4.2.4 Table Protection

6.4.2.4.1 General

Table Protection is an optional box containing protection information associated with a Table to support confidentiality (encryption), integrity verification (digital signature) and access control policy enforcement on selected regions of the Table as required by the user.

6.4.2.4.2 Syntax

Table 11 – Table Protection syntax

Syntax	Key	Type	Remarks
table_protection {	*tbpr*		
dataset_group_ID		u(8)	
dataset_ID		u(16)	
table_ID		u(8)	
TB_protection_value()			
}			

6.4.2.4.3 Semantics

TB_protection_value() contains compressed protection metadata. The decoding process to retrieve the XML document from the coded representation is specified in Clause 9 of Part 3. It consists of three main components: encryption parameters, privacy policy and digital signatures. Details on the XML schema for table protection metadata will be provided in a separate document.

Controlled access to and authentication of data subsets within a table are enabled through privacy rules and signature elements in the schema. Like the protection metadata at the Dataset Group and Dataset levels, the privacy rules specify who can execute a given action and under which conditions, and the information is conveyed according to the eXtensible Access Control Markup Language (XACML) Version 3.0 specification. Users may define the attributes, chunks, genomic regions, and ranges of table indices on which a privacy rule is applied. Any number of XML signature elements can be present in the Table Protection box and shall use a URI to specify the attributes and chunks associated with each signature. Detached, Enveloped and Enveloping signatures are supported. If decryption is required, signature verification shall be performed before decryption.

6.4.3 Table Data

6.4.3.1 General

Table Data is a container box that allows Table attributes to be grouped and organized by their roles as: main data, auxiliary data associated with the rows/columns of the main data, auxiliary row/column linkages with other Tables, and any other auxiliary data as defined by users.

There are two ways to organize data payloads in Table Data:

— If either attribute_dependent_chunks or attribute_contiguity equals 1, group data payloads into Table Blocks by attribute (block_type == 1) and order them by chunk as in the corresponding chunk structure in Table Data Master Index.

— Otherwise, group data payloads into Table Blocks by chunk (block_type == 0) and order them by attribute as in Table Data Attribute Information. Chunk contiguity is only allowed when the same chunk structure is shared among all attributes and attribute_contiguity is set to 0.

6.4.3.2 Syntax

Table 12 – Table Data syntax

Syntax	Key	Type	Remarks
table_data {	*tdcn*		
table_data_ID		u(3)	
table_data_class		u(3)	
two_dimensional		u(1)	
column_major_chunk_order		u(1)	
for (i=0; i<(two_dimensional + 1); i++)			
dimension_size[i]		u(table_index_size*8)	
attribute_info	tdai	gen_info	As specified in 6.4.4
master_index	tdmi	gen_info	As specified in 6.4.5
supplementary_indices	tdsi	gen_info	As specified in 6.4.6
if (!attribute_dependent_chunks && !attribute_contiguity) {			Parameters defined in Table 16
block_type = 0			Assigned to Table Data Blocks indicating chunk contiguity
if (variable_size_chunks || !two_dimensional) {			variable_size_chunks defined in the uniform chunk structure in Table 16
for (i=0; i<n_chunks; i++) {			n_chunks defined in the uniform chunk structure in Table 16
if (sizeof(payload_chunk[i]) > 0)			Omit the block for the chunk of indices (i, j) if payload is empty
table_block_by_chunk[i]	tdbl	gen_info	As specified in 6.4.7
}			
}			
else if (column_major_chunk_order) {			
for (j=0; j<n_chunks_per_row; j++) {			The number of chunks per row/ column is either inferred from

for (i=0; i<n_chunks_per_col; i++) {			dimension_size and chunk_size, or the range of chunk indices of the transported Table Data Blocks.
if (sizeof(payload_chunk[i][j]) > 0)			
table_block_by_chunk[i][j]	tbdl	gen_info	As specified in 6.4.7
}			
}			
}			
else {			
for (i=0; i<n_chunks_per_col; i++) {			
for (j=0; j<n_chunks_per_row; j++) {			
if (sizeof(payload_chunk[i][j]) > 0)			
table_block_by_chunk[i][j]	tbdl	gen_info	As specified in 6.4.7
}			
}			
}			
}			
else {			
block_type = 1			Assigned to Table Data Blocks indicating attribute contiguity
for (i=0; i<n_attributes; i++) {			
if (sizeof(payload_attribute[i] > 0)			Omit the block for the i-th attribute if payload is empty
table_block_by_attribute[i]	tdbl	gen_info	As specified in 6.4.7
}			
}			
}			

6.4.3.3 Semantics

table_data_ID is the unique identifier of the Table Data within the Table.

table_data_class specifies the class of the Table Data. The possible values are:

— 0 – main Table Data

— 1 – auxiliary Table Data for data attributes mapped to the rows of the main Table Data

— 2 – auxiliary Table Data for data attributes mapped to the columns of the main Table Data

— 3 – auxiliary Table Data for linkage attributes mapped to the rows of the main Table Data

— 4 – auxiliary Table Data for linkage attributes mapped to the columns of the main Table Data

— 5-7 – any auxiliary Table Data defined by the user

Within a Table, there can only be one main Table Data of class 0. For auxiliary Table Data of classes 1-4, if the attributes are two-dimensional, the mapping is always between the rows of the auxiliary Table Data and the rows (classes 1 and 3) or columns (classes 2 and 4) of the main Table Data.

two_dimensional is a flag, and if set to 1, indicates that all attributes in the Table Data are 2-d. Otherwise, all attributes in the Table Data are 1-d. For main Table Data, its value should be the same as two_dimensional_main.

column_major_chunk_order is a flag only relevant for an attribute when two_dimensional == 1 and variable_size_chunks == 0 in the corresponding Table Data Chunk Structure. If set to 1, it indicates that the chunks of the attribute within the Table Data Block (block_type == 1) are in column-major order. Otherwise, the chunks are in row-major order.

dimension_size[i] specifies the total number of rows (i == 0) or columns (i == 1) in the Table Data when two_dimensional == 1, or simply the number of elements when two_dimensional == 0. In the case of transport when data generation is ongoing, a value of 0 can be applied. At the completion of data generation and transport, the value(s) of dimension_size[] should be computed and reassigned.

6.4.4 Table Data Attribute Information

6.4.4.1 General

Table Data Attribute Information is a collection of attribute definitions encapsulated in attribute_parameter_set, with the number of attributes specified in n_attributes.

Table 13 – Table Data Attribute Information syntax

Syntax	Key	Type	Remarks
table_data_attribute_information {	*tdai*		
table_data_header		table_data_header	As specified in 6.6.4
n_attributes		u(16)	
for (i=0; i<n_attributes; i++)			
attribute_parameter_set[i]	tdap	gen_info	As specified in 6.4.4.2
}			

6.4.4.2 Table Data Attribute Parameter Set

6.4.4.2.1 General

Table Data Attribute Parameter Set is a box that contains the definitions of an attribute, including some basic information and configurations of its associated compressor.

6.4.4.2.2 Syntax

Table 14 – Table 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 defined in

			compressor in Table 7
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	
}			

6.4.4.2.3 Semantics

attribute_ID is the identifier of the attribute unique within Table Data. It is the same as the index of the attribute in attribute_parameter_set of Table Data Attribute Information.

attribute_name is the name of the attribute.

attribute_metadata is the metadata of the attribute, which can include a description on the meaning and format of the attribute value and its belonging attribute group.

attribute_type specifies the data type of the attribute. The possible values and their respective data type definitions are listed in Table 15.

Table 15 – Attribute type definitions

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

attribute_default_value is the default value of the attribute, mainly used for sparse encoding when most values equal to the default are excluded.

attribute_missing_value is the missing value of the attribute to be used in place of a null value in the output after decompression.

compressor_ID is the ID of one of the compressors defined in Dataset Parameter Set for compressing the data of the attribute.

(**dependency_table_data_ID[i]**, **dependency_attribute_ID[i]**) correspond to the table ID and and attribute ID of the i-th dependency attribute required by the transform algorithm (if transform == 1) within the compressor referenced by compressor_ID.

compressor_common_data stores the codebooks/statistical models used by the associated compressor to apply commonly on all chunks.

Attribute configurations for different genomic file types are specified in document M53383 "Philips' Response to CE2 of MPEG-G Part 6".

6.4.5 Table Data Master Index

6.4.5.1 General

Table Data Master Index is a container box of indexing information that includes the definition of chunk structure(s), i.e. the range of indices (both rows and columns for 2-d data) per chunk, and byte-offset pointers to individual Table Blocks and their subsidiary payloads.

Table 16 – Table Data Master Index syntax

Syntax	Key	Type	Remarks
table_data_master_index {	*tdmi*		
table_data_header		table_data_header	As specified in 6.6.4
reserved		u(6)	
attribute_dependent_chunks		u(1)	
attribute_contiguity		u(1)	
if (!attribute_dependent_chunks)			
chunk_structure	tdcs	gen_info	As specified in 6.4.5.2
else {			
for (i=0; i<n_attributes; i++)			
chunk_structure[i]	tdcs	gen_info	As specified in 6.4.5.2
}			
payload_byte_offset	tdbo	gen_info	As specified in 6.5.2
}			

attribute_dependent_chunks is a flag, and if set to 1, indicates that each attribute has a different chunk structure. Otherwise, all attributes share the same chunk structure.

attribute_contiguity is a flag, and if set to 1, indicates that the data payloads are grouped into Table Blocks by attribute. Otherwise, data payloads are grouped into Table Blocks by chunk.

6.4.5.2 Table Data Chunk Structure

6.4.5.2.1 General

Table Data Chunk Structure is a box specifying how the 1-d or 2-d attribute data should be divided into rectangular chunks defined by ranges of row and column indices.

6.4.5.2.2 Syntax

Table 17 – Table Data Chunk Structure syntax

Syntax	Key	Type	Remarks
table_data_chunk_structure {	*tdcs*		

reserved		u(7)	
variable_size_chunks		u(1)	
n_chunks		u(table_index_size*8)	
if (variable_size_chunks) {			
for (i=0; i<n_chunks; i++) {			
for (j=0; j<(two_dimensional +1); j++) {			two_dimensional defined in Table 12
start_index[i][j]		u(table_index_size*8)	
end_index[i][j]		u(table_index_size*8)	
}			
}			
}			
else {			
for (j=0; j<(two_dimensional + 1); j++)			
chunk_size[j]		u(table_index_size*8)	
}			
}			

6.4.5.2.3 Semantics

variable_size_chunks is a flag, and if set to 1, indicates that the size of each chunk is different, and thus the corresponding start and end indices are specified independently. Otherwise, a uniform size applies to all chunks. If the number of rows/columns is unknown as in the case of data generation and transport, a uniform chunk size should be applied with variable_size_chunks set to 0.

n_chunks specifies the total number of chunks defined in this chunk structure. In the case of transport when data generation is ongoing, a value of 0 can be applied if the number is unknown. At the completion of data generation and transport, the value of n_chunks should be computed and reassigned. The number of bits for n_chunks is the same as the number of bits for column/row index, i.e. table_index_size*8, to allow having one chunk per row or column.

(start_index[i][j], end_index[i][j]) is the pair of start and end indices defining the range of rows (j == 0) or columns (j == 1) for the i^{th} rectangular chunk, only used when variable_size_chunks == 1.

chunk_size[j] specifies the number of rows (j == 0) or columns (j == 1) per chunk, only used when variable_size_chunks == 0.

6.4.6 Table Data Supplementary Indices

6.4.6.1 General

Table Data Supplementary Indices is an optional container box that carries additional attribute-specific indexing data for enabling query search based on criteria such as genomic region, gene symbol or any other attributes.

Table 18 – Table Data Supplementary Indices syntax

Syntax	Key	Type	Remarks
table_data_supplementary_indices {	*tdsi*		
table_data_header		table_data_header	As specified in 6.6.4
n_supp_indices		u(8)	
for (i=0; i<n_supp_indices; i++)			
supp_index_data[i]	tdsd		As specified in 6.4.6.2

}			

n_supp_indices specifies the number of supplementary indices associated with the Table Data.

6.4.6.2 Table Data Supplementary Index Data

6.4.6.2.1 General

Table Data Supplementary Index Data is a box containing information and data of a supplementary index.

6.4.6.2.2 Syntax

Table 19 – Table Data Supplementary Index Data syntax

Syntax	Key	Type	Remarks
table_data_supplementary_index_data {	*tdsd*		
n_index_attributes		u(8)	
for (i=0; i<n_index_attributes; i++)			
index_attribute_ID[i]		u(16)	
index_type		st(v)	
index_data		u(index_data_size*8)	
}			

6.4.6.2.3 Semantics

n_index_attributes is the number of attributes associated with the supplementary index.

index_attribute_ID is the ID of an attribute within the same Table Data associated with the supplementary index.

index_type specifies the type of the supplementary index. Possible values include “CSI” (Crowd Sourced Indexing), “B-Tree”, “R-Trees” and “LevelDB”.

index_data is the indexing data on which queries by attribute values are performed to return the row and/or column indices of the matched data. The size of index_data is given by index_data_size = Length – [13 + n_index_attributes × 2 + sizeof(index_type)], where Length is defined in the gen_info header of the tdsd container.

6.4.7 Table Data Block

6.4.7.1 General

Table Data Block is a box containing the compressed payloads, either of the same chunk and ordered by attributes (block_type == 0 for chunk contiguity), or of the same attribute and ordered by chunks (block_type == 1 for attribute contiguity).

6.4.7.2 Syntax

Table 20 – Table Data Block syntax

Syntax	Key	Type	Remarks
table_data_block {	*tdbl*		

table_data_header		table_data_header	As specified in 6.6.4
reserved		u(7)	
block_type		u(1)	
if (block_type == 0) {			
if (variable_size_chunks || !two_dimensional) {			variable_size_chunks defined in the uniform chunk structure in Table 10; two_dimensional defined in Table 12
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++) {			n_attributes defined in Table 13
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) {			column_major_chunk_order defined in Table 12
n_chunks_per_col		u(table_index_size*8)	
for (k=0; k<n_chunks_per_row; k++) {			The number of chunks per row/column is determined by the total number of columns/ rows in the Table Data and the row/column chunk size of the attribute.
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_ind	

		ex_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)	
}			
}			
}			
}			
}			

6.4.7.3 Semantics

block_type is the type of the Table Data Block. The possible values are: 0 = chunk-contiguous (consisting of payloads of different attributes belonging to the same chunk) and 1 = attribute-contiguous (consisting of payloads of different chunks belonging to the same attribute).

(**chunk_idx_1**, **chunk_idx_2**) is the pair of row and column indices of the chunk associated with the Table Data Block, only applicable when block_type == 0 (chunk-contiguous and implying same chunk structure across all attributes). When the Table Data is 2-d (two_dimensional == 1) and a fixed chunk size is applied (variable_size_chunks == 0), the pair of indices starts from (0, 0) at the top-left of the Table Data, and increases by 1 for the next chunk towards the right/bottom. When the Table Data is 1-d (two_dimensional == 0) or the chunk size is variable (variable_size_chunks == 1), only chunk_idx_1 is used and chunk_idx_2 is set to 0.

attribute_ID is the index of the attribute associated with the Table Data Block, only applicable when block_type == 1 (attribute-contiguous). The attribute index, counting from 0, should be in the same order as the array of attribute_parameter_set in Table Data Attribute Information.

n_chunks_per_col specifies the total number of chunks in a column, only used when block_type == 1 (attribute-contiguous), variable_size_chunks == 0, two_dimensional == 1 and column_major_chunk_order == 1. This number is needed for computing the row and column indices of each chunk in the 2-d Table Data for data access and reconstruction.

n_chunks_per_row specifies the total number of chunks in a row, only used when block_type == 1 (attribute-contiguous), variable_size_chunks == 0, two_dimensional == 1 and column_major_chunk_order == 0. This number is needed for computing the row and column indices of each chunk in the 2-d Table Data for data access and reconstruction.

Note that the values of n_chunks, n_chunks_per_column and n_chunks_per_row are specific to the attribute referred to by attribute_ID if attribute_dependent_chunks == 1.

(**payload_size[i][j][k]**, **payload[i][j][k]**) are the size in number of bytes and data of the compressed payload that corresponds to the chunk of row and column indices (j, k) in the i-th attribute. Note that even for empty payloads, payload_size must be included and set to 0.

6.5 Data structures specific to file format

6.5.1 General

This subclause specifies the data structures specific to the storage of genomic information, in addition to the data structures specified in subclause 6.4.

6.5.2 Table Data Byte Offset

6.5.2.1 General

Table Data Byte Offset is a box containing the byte-offset pointers to the Table Data Blocks and their individual payloads.

6.5.2.2 Syntax

Table 21 – Table Data Byte Offset syntax

Syntax	Key	Type	Remarks
table_data_byte_offset {	*tdbo*		
if (!attribute_dependent_chunks && !attribute_contiguity) {			Parameters defined in Table 16
if (variable_size_chunks || !two_dimensional) {			variable_size_chunks defined in the uniform chunk structure in Table 10; two_dimensional defined in Table 12
for (j=0; j<n_chunks; j++) {			n_chunks defined in the uniform chunk structure in Table 10
chunk_block_offset[j]		u(byteOffsetSize)	
if (chunk_block_offset[i] > 0) {			If Table Data Block for chunk i exists.
for (i=0; i<n_attributes; i++)			n_attributes defined in Table 13
payload_offset[i][j]		u(byteOffsetSize)	
}			
}			
}			
else if (column_major_chunk_order) {			column_major_chunk_order defined in Table 12
for (k=0; k<n_chunks_per_row; k++) {			The number of chunks per row/column is either inferred from dimension_size and chunk_size, or the range of chunk indices of the transported Table Data Blocks.
for (j=0; j<n_chunks_per_col; j++) {			
chunk_block_offset[j][k]		u(byteOffsetSize)	
if (chunk_block_offset[j][k] > 0)			If Table Data Block for chunk (j, k) exists
for (i=0; i<n_attributes; i++)			
payload_offset[i][j][k]		u(byteOffsetSize)	
}			
}			
}			
}			

else {			
for (j=0; j<n_chunks_per_col; j++) {			
for (k=0; k<n_chunks_per_row; k++) {			
chunk_block_offset[j][k]		u(byteOffsetSize)	
if (chunk_block_offset[j][k] > 0) {			
for (i=0; i<n_attributes; i++)			
payload_offset[i][j][k]		u(byteOffsetSize)	
}			
}			
}			
}			
}			
else {			
for (i=0; i<n_attributes; i++) {			
attribute_block_offset[i]		u(byteOffsetSize)	
if (attribute_block_offset[i] > 0) {			If Table Data Block for Attribute i exists
if (variable_size_chunks[i] || !two_dimensional) {			
for (j=0; j<n_chunks[i]; j++)			
payload_offset[i][j]		u(byteOffsetSize)	
}			
else if (column_major_chunk_order) {			
for (k=0; k<n_chunks_per_row[i]; k++) {			
for (j=0; j<n_chunks_per_col[i]; j++)			
payload_offset[i][j][k]		u(byteOffsetSize)	
}			
}			
else {			
for (j=0; j<n_chunks_per_col[i]; j++) {			
for (k=0; k<n_chunks_per_row[i]; k++)			
payload_offset[i][j][k]		u(byteOffsetSize)	
}			
}			
}			
}			
}			

6.5.2.3 Semantics

chunk_block_offset[j][k] is the byte offset, counting from the beginning of the associated Table Data container, to a chunk-contiguous Table Data Block (block_type == 0) that contains the payload data of all attributes for the chunk of row and column indices (j, k). Its value should be 0 if the Table Data Block for chunk (j, k) does not exist when the payloads are all empty. If variable_size_chunks == 1 and two_dimensional == 0, the second index [k] can be dropped.

attribute_block_offset[i] is the byte offset, counting from the beginning of the associated Table Data container, to an attribute-contiguous Table Data Block (block_type == 1) that contains the payload data of all chunks for the i-th attribute as defined in Table Data Attribute Information. Its value should be 0 if the Table Data Block for the i-th attribute does not exist when the payloads are all empty.

payload_offset[i][j][k] is the byte offset, counting from the beginning of the encapsulating Table Data Block container, to the compressed payload data that corresponds to the chunk of row and column indices (j, k) in the i-th attribute. Note that even for empty payloads, payload_size must be included and set to 0. If variable_size_chunks == 1 and two_dimensional == 0, the third index [k] can be dropped.

Note that if attribute_dependent_chunks == 1, the values of n_chunks[i], n_chunks_per_row[i] and n_chunks_per_col[i] are specific to the i-th attribute. Otherwise, their values are uniform across all attributes and the index [i] can be dropped.

6.6 Data structures specific to transport format

6.6.1 General

This subclause specifies the data structures specific to the transport of genomic information, in addition to the data structures specified in subclause 6.4.

6.6.2 Data Streams

A data stream is identified by a unique Stream_ID, equal to the SID field of packet header as specified in subclause 6.7.5.2 of Part 1, and it can transport any of the following data structures:

— File Header, as specified in subclause 6.5.1 of Part 1: this data stream shall be unique and composed by one or more packets with Stream ID (SID in packet header, as specified in subclause 6.7.5.2 of Part 1) equal to 1.

— Dataset Group Header, as specified in subclause 6.5.2.2 of Part 1,

— Dataset Header, as specified in subclause 6.4.1.2,

— Dataset Parameter Set, as specified in subclause 6.4.1.3,

— Table Header, as specified in subclause 6.4.2.2,

— Table Data Attribute Information, as specified in subclause 6.4.4,

— Table Data Master Index, as specified in subclause 6.4.5,

— Table Data Supplementary Indices, as specified in subclause 6.4.6,

— Table Data Block, as specified in subclause 6.4.7,

— data structures containing transport information (dataset mapping table list as specified in subclause 6.7.3 of Part 1, and dataset mapping table as specified in subclause 6.6.3),

— metadata and protection information, as specified in subclauses 6.5.2.6, 6.5.2.7, 6.5.3.3 and 6.5.3.4 of Part 1, and subclauses 6.4.2.3 and 6.4.2.4.

6.6.3 Dataset Mapping Table

6.6.3.1 General

Dataset Mapping Table is a mandatory box listing all data streams transporting data related to the dataset identified by dataset_ID. The syntax and semantics of Dataset Mapping Table remain the same as described in subclause 6.7.4 of Part 1. To support the transport of the new data structures specific to annotation datasets, Table 37 of Part 1 is extended to include new data types that identify the types of data structure carried by packets. Table 16 shows the list of data types relevant to annotation datasets, with those already existing in Part 1 in gray text.

Table 22 – data_type field semantics

data_type	Data structure	Subclause
0	dataset_group_header	6.5.2.2 of Part 1
3	dataset_header	6.5.3.2 of Part 1
4	dataset_parameter_set	6.4.1.3
5	dataset_group_metadata	6.5.2.6 of Part 1
6	dataset_metadata	6.5.3.3 of Part 1
7	dataset_group_protection	6.5.2.7 of Part 1
8	dataset_protection	6.5.3.4 of Part 1
15	table_header	6.4.2.2
16	table_data_attribute_information	6.4.4
17	table_data_master_index	6.4.5
18	table_data_supplementary_indices	6.4.6
19	table_data_block	6.4.7
20	table_metadata	6.4.2.3
21	table_protection	6.4.2.4

For the Table-related data types 15-21, it is recommended that their associated data_SID (Data Stream ID), as specified in subclause 6.7.4 of Part 1, to be unique across different dataset_ID and dataset_group_ID for ease of implementation. However, the same data_SID can also be reused by the same data type of different dataset_ID and dataset_group_ID, provided the data structures are transported one after another in the same stream without interleaving of their packets, since the associated dataset_ID and dataset_group_ID are carried within the data structures. If data is generated by parallel processes, more than one data_SID can be assigned to data type 19 to speed up the transmission of Table Data Blocks, which carry the table payloads.

6.6.4 Table Data Header

6.6.4.1 General

Table Data Header is a mandatory data structure in the transport format for the four boxes – Table Data Attribute Information, Table Data Master Index, Table Data Supplementary Indices and Table Data Block – under Table Data. It contains the IDs of the upper-level containers that are required for the assembly of the Table Data structures after transport, but is excluded from the file format.

6.6.4.2 Syntax

Table 23 – Table Data Header syntax

Syntax	Key	Type	Remarks

table_data_header {			
reserved		u(5)	
dataset_group_ID		u(8)	
dataset_ID		u(16)	
table_ID		u(8)	
table_data_ID		u(3)	
}			

6.6.4.3 Semantics

dataset_group_ID is the identifier of dataset group containing the dataset identified by dataset_ID.

dataset_ID is the identifier of the dataset containing the Table identified by table_ID.

table_ID is the identifier of the table containing the Table Data identified by table_data_ID.

table_data_ID is the identifier of the Table Data containing the data structures associated with this Table Data Header.

7 Strengths and features

7.1 Compatibility with ISO/IEC 23092 (MPEG-G) Series

This file and transport format for genomic annotation data is fully compatible with other MPEG-G parts in the following ways:

— The overall data structures and hierarchical encapsulation levels (subclause 6.1.1 of Part 1), general syntax and semantics (subclause 6.2 of Part 1), and the gen_info box structure (subclause 6.3 of Part 1) are preserved.

— The syntax and usage of the highest-level containers File and Dataset Group remain the same as specified in Part 1.

— The syntax of the Dataset container in Part 1 is extended to support a new Dataset Type specific to the representation of genomic annotation data.

— The Dataset Parameter Set container in Part 1 is extended to store the parameters, mainly compressor definitions, used by annotation datasets.

— The coding/decoding process for the metadata and protection fields in Table is the same as the metadata and protection fields in Dataset Group and Dataset, with the XML schema extended to carry information specific to Tables and support new functionalities.

— The data transport mechanism described in Part 1 remains the same, with Dataset Mapping Table extended to support new data types that correspond to container boxes specific to annotation datasets.

7.2 Interoperability with existing MPEG-G components

Among the list of available decoding software components specified in subclause 6.3.2 of Part 4, this format should be interoperable with core decoder – decapsulator, parameter set parser and

CABAC engine. Most of the other components are specific to the decoding of sequencing data and are therefore not applicable to annotation datasets.

7.3 Indexing capabilities

Indexing capabilities are realized through the following data structures:

— Table Data Master Index that provides the mapping between row and/or column indices of a table and specific chunks of an attribute.

— Table Data Supplementary Indices that provides the mapping between row and/or column indices of a table and values of selected attributes such as genomic position and gene symbols.

Compound query that consists of a logical combination of attribute conditions can be realized by (1) looking up the row and/or column indices satisfying each attribute condition independently, (2) identifying the subset of indices satisfying the logics in the compound query, (3) mapping the subset of indices to specific chunks of an attribute, and (4) looking up the locations of the payloads of the matching chunks.

7.4 Selective access to data subsets

Selective access is enabled through data chunking, i.e. dividing each table attribute into rectangular chunks, which are then compressed individually. To access data in specific regions of the table, the chunks in those regions are identified and their payloads located using the information in Table Data Master Index. Decompression is then applied only on the payloads of the matching chunks to retrieve the original data in the requested regions.

7.5 Controlled access to and authentication of data subsets

Controlled access to and authentication of data subsets within a table in an annotation dataset are enabled through privacy rules and signature elements in the schema for Table Protection metadata. Users may define the attributes, chunks, genomic regions, and ranges of table indices on which a privacy rule is applied. There can be any number of XML signature elements in protection metadata and a URI should be used to specify the attributes and chunks associated with each signature.

7.6 Data linkages

This format supports the creation of data linkages useful for join table query and efficient data visualization. Data linkages can be defined as URIs in:

— Table Metadata – at this level, linkages can be between two datasets, e.g. an annotation table and its originating sequencing dataset, or two tables, where the rows/columns of one table are mapped to the rows/columns of another table

— Auxiliary Table Data (classes 3 and 4) for row/column linkage attributes – at this level, a linkage is defined per row/column. For example, in a VCF file, each sample in the column should be linked to its corresponding sequencing dataset from which its variants are called.

7.7 Simplicity of syntax

Since this format for genomic annotation data is fully integrated into the MPEG-G container box hierarchy and uses the same data structures for transport, it can keep the syntax succinct by introducing only data structures specific to the organization of data within a table.

7.8 Flexibility

Flexibility is one of the main design principles of this format and is offered in the following aspects:

— Customizable compressor configurations for adopting new transform and compression algorithms

— Customizable attribute definitions for accommodating new annotation file types

— Multiple Tables can be stored in a dataset, e.g. to represent data at different resolutions

— Multiple auxiliary Table Data can be provided to supplement the main Table Data with additional information

— Flexible chunk structure – uniform or attribute-dependent, fixed- or variable-size – for optimum compression and random access performance

— Attribute or chunk contiguity in grouping payloads into Table Data Blocks

— Row- or column-major chunk order in organizing payloads or Table Data Blocks

— Different symmetry modes (0 = unsymmetrical; 1 = symmetrical; 2 = skew-symmetric; 3 = Hermitian; 4-7 = reserved or user-defined) over the major/minor diagonal of a matrix

7.9 Support for future extensions

This format can be readily extended to accommodate any future annotation file types and transform/compression algorithms by providing new configurations of attributes and compressors defined in Table Data Attribute Information and Dataset Parameter Set.