INTERNATIONAL STANDARD

ISO/IEC 23092-1

Second edition
2020-10

Information technology — Genomic information representation —

Part 1:
Transport and storage of genomic information

Technologie de l'information — Représentation des informations génomiques —

Partie 1: Transport et stockage des informations génomiques

Reference number
ISO/IEC 23092-1:2020(E)

ISO copyright office
CP 401 • Ch. de Blandonnet 8
CH-1214 Vernier, Geneva
Phone: +41 22 749 01 11
Email: copyright@iso.org
Website: www.iso.org

Published in Switzerland

Contents

Foreword

ISO (the International Organization for Standardization) and IEC (the International Electrotechnical Commission) form the specialized system for worldwide standardization. National bodies that are members of ISO or IEC participate in the development of International Standards through technical committees established by the respective organization to deal with particular fields of technical activity. ISO and IEC technical committees collaborate in fields of mutual interest. Other international organizations, governmental and non-governmental, in liaison with ISO and IEC, also take part in the work.

The procedures used to develop this document and those intended for its further maintenance are described in the ISO/IEC Directives, Part 1. In particular, the different approval criteria needed for the different types of document should be noted. This document was drafted in accordance with the editorial rules of the ISO/IEC Directives, Part 2 (see www.iso.org/directives).

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. ISO and IEC shall not be held responsible for identifying any or all such patent rights. Details of any patent rights identified during the development of the document will be in the Introduction and/or on the ISO list of patent declarations received (see www.iso.org/patents) or the IEC list of patent declarations received (see http://patents.iec.ch).

Any trade name used in this document is information given for the convenience of users and does not constitute an endorsement.

For an explanation of the voluntary nature of standards, the meaning of ISO specific terms and expressions related to conformity assessment, as well as information about ISO's adherence to the World Trade Organization (WTO) principles in the Technical Barriers to Trade (TBT) see www.iso.org/iso/foreword.html.

This document was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee SC 29, *Coding of audio, picture, multimedia and hypermedia information*.

This second edition cancels and replaces the first edition (ISO/IEC 23092-1:2019), which has been technically revised.

The main changes compared to the previous edition are as follows:

- reference box syntax and semantics have been updated;
- syntax, semantics and decoding process for cluster signatures has been fixed;
- the scope of some parameters has been changed from dataset_header to dataset_parameter_set;
- new dataset_group_ID and dataset_ID fields have been added to the metadata and protection boxes;
- minor fixes in transport format;
- editorial changes.

A list of all parts in the ISO/IEC 23092 series can be found on the ISO website.

Any feedback or questions on this document should be directed to the user's national standards body. A complete listing of these bodies can be found at www.iso.org/members.html.

Introduction

The advent of high-throughput sequencing (HTS) technologies has the potential to boost the adoption of genomic information in everyday practice, ranging from biological research to personalized genomic medicine in clinics. As a consequence, the volume of generated data has increased dramatically during the last few years, and an even more pronounced growth is expected in the near future.

At the moment, genomic information is mostly exchanged through a variety of data formats, such as FASTA/FASTQ for unaligned sequencing reads and SAM/BAM/CRAM for aligned reads. With respect to such formats, the ISO/IEC 23092 series provides a new solution for the representation and compression of genome sequencing information by:

- Specifying an abstract representation of the sequencing data rather than a specific format with its direct implementation.
- Being designed at a time point when technologies and use cases are more mature. This permits addressing one limitation of the textual SAM format, for which the incremental ad-hoc addition of features followed along the years, resulting in an overall redundant and suboptimal format which was unnecessarily complicated.
- Separating free-field user-defined information with no clear semantics from the genomic data representation. This allows a fully interoperable and automatic exchange of information between different data producers.
- Allowing multiplexing of relevant metadata information with the data since data and metadata are partitioned at different conceptual levels.
- Following a strict and supervised development process which has proven successful in the last 30 years in the domain of digital media for the transport format, the file format, the compressed representation and the application program interfaces.

The ISO/IEC 23092 series provides the enabling technology that will allow the community to create an ecosystem of novel, interoperable, solutions in the field of genomic information processing. In particular it offers:

- Consistent, general and properly designed format definitions and data structures to store sequencing and alignment information. A robust framework which can be used as a foundation to implement different compression algorithms.
- Speed and flexibility in the selective access to coded data, by means of newly-designed data clustering and optimized storage methodologies.
- Low latency in data transmission and consequent fast availability at remote locations, based on transmission protocols inspired by real-time application domains.
- Built-in privacy and protection of sensitive information, thanks to a flexible framework which allows customizable, secured access at all layers of the data hierarchy.
- Reliability of the technology and interoperability among tools and systems, owing to the provision of a procedure to assess conformance to this document on an exhaustive dataset.
- Support to the implementation of a complete ecosystem of compliant devices and applications, through the availability of a normative reference implementation covering the totality of the ISO/IEC 23092 series.

The fundamental structure of the ISO/IEC 23092 series data representation is the *genomic record*. The genomic record is a data structure consisting of either a single sequence read, or a paired sequence read, and its associated sequencing and alignment information; it may contain detailed mapping and alignment data, a single or paired read identifier (read name) and quality values.

Without breaking traditional approaches, the genomic record introduced in the ISO/IEC 23092 series provides a more compact, simpler and manageable data structure grouping all the information related to a single DNA template, from simple sequencing data to sophisticated alignment information.

The genomic record, although it is an appropriate logic data structure for interaction and manipulation of coded information, is not a suitable atomic data structure for compression. To achieve high compression ratios, it is necessary to group genomic records into clusters and to transform the information of the same type into sets of descriptors structured into homogeneous blocks. Furthermore, when dealing with selective data access, the genomic record is a too small unit to allow effective and fast information retrieval.

For these reasons, this document introduces the concept of access unit, which is the fundamental structure for coding and access to information in the compressed domain.

The access unit is the smallest data structure that can be decoded by a decoder compliant with ISO/IEC 23092-2. An access unit is composed of one block for each descriptor used to represent the information of its genomic records; therefore, a block payload is the coded representation of all the data of the same type (i.e. a descriptor) in a cluster.

In addition to clusters of genomic records compressed into access units, reads are further classified in six data classes: five classes are defined according to the result of their alignment against one or more reference sequences; the sixth class contains either reads that could not be mapped or raw sequencing data. The classification of sequence reads into classes enables the development of powerful selective data access. In fact, access units inherit a specific data characterization (e.g. perfect matches in Class P, substitutions in Class M, indels in Class I, half-mapped reads in Class HM) from the genomic records composing them, and thus constitute a data structure capable of providing powerful filtering capability for the efficient support of many different use cases.

Access units are the fundamental, finest grain data structure in terms of content protection and in terms of metadata association. In other words, each access unit can be protected individually and independently. Figure 1 shows how access units, blocks and genomic records relate to each other in the ISO/IEC 23092 series data structure.

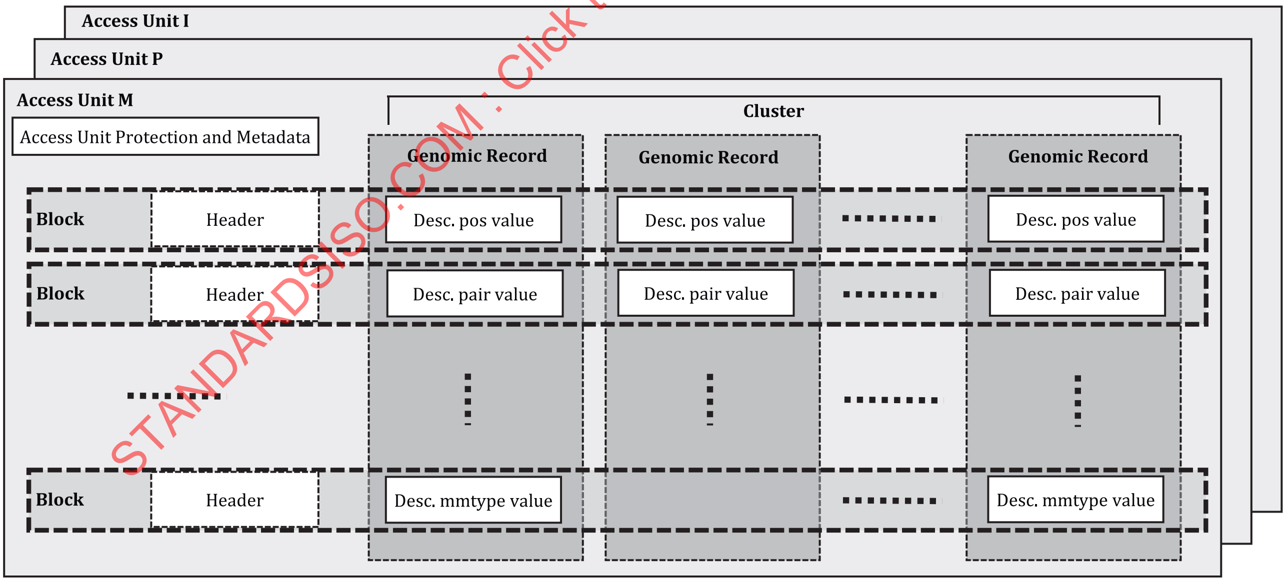

Figure 1 — Access units, blocks and genomic records

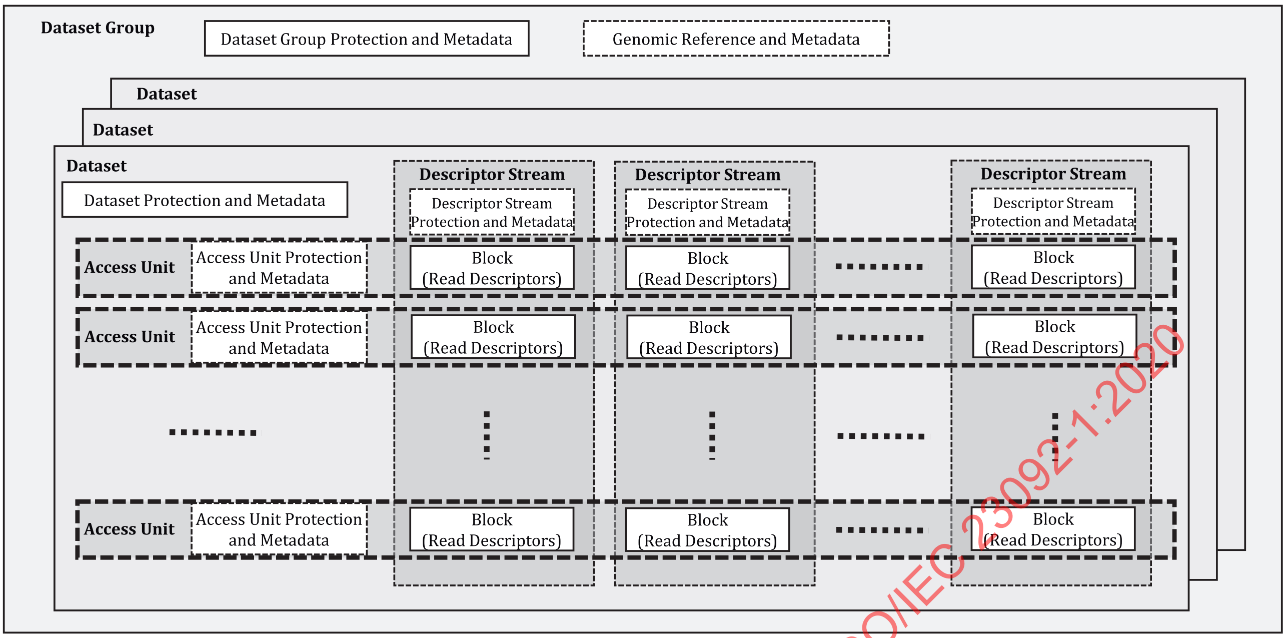

Figure 2 — High-level data structure: datasets and dataset group

A dataset is a coded data structure containing headers and one or more access units. Typical datasets could, for example, contain the complete sequencing of an individual, or a portion of it. Other datasets could contain, for example, a reference genome or a subset of its chromosomes. Datasets are grouped in dataset groups, as shown in Figure 2.

A simplified diagram of the dataset decoding process is shown in Figure 3.

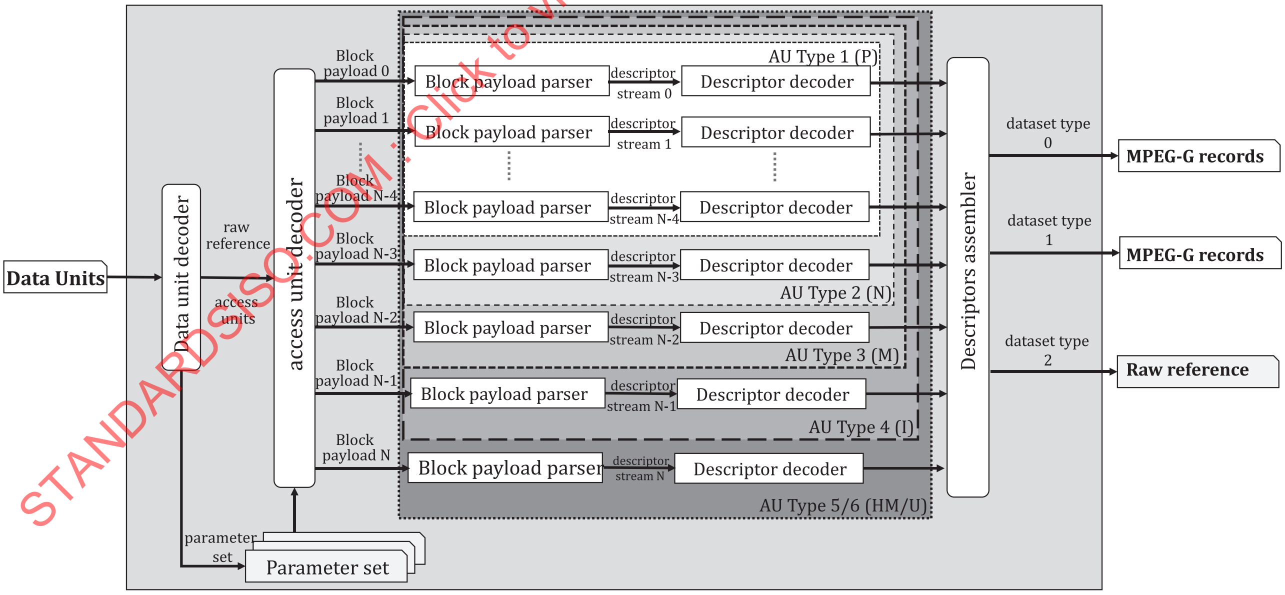

Figure 3 — Decoding process

This document defines the syntax and semantics of the data formats for both transport and storage of genomic information. According to this document, the compressed sequencing data can be multiplexed into a bitstream suitable for packetization for real-time transport over typical network protocols. In storage use cases, coded data can be encapsulated into a file format with the possibility to organize blocks per descriptor stream or per access units, to further optimize the selective access performance to the type of data access required by the different application scenarios. This document further provides a reference process to convert a transport stream into a file format and vice versa.

The International Organization for Standardization (ISO) and International Electrotechnical Commission (IEC) draw attention to the fact that it is claimed that compliance with this document may involve the use of a patent.

ISO and IEC take no position concerning the evidence, validity and scope of this patent right.

The holder of this patent right has assured ISO and IEC that he/she is willing to negotiate licences under reasonable and non-discriminatory terms and conditions with applicants throughout the world. In this respect, the statement of the holder of this patent right is registered with ISO and IEC. Information may be obtained from the patent database available at www.iso.org/patents.

Information technology — Genomic information representation —

Part 1:
Transport and storage of genomic information

1 Scope

This document specifies data formats for both transport and storage of genomic information, including the conversion process.

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 23092-2, *Information technology — Genomic information representation — Part 2: Coding of genomic information*

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

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
access unit
logical data structure containing a coded representation of genomic information to facilitate bit stream access and manipulation

3.2
access unit covered region
genomic range comprised between the access unit start position and the access unit end position, inclusive

3.3
access unit start position
position of the left-most mapped base among the first alignments of all genomic records contained in the access unit, irrespective of the strand

3.4
access unit end position
position of the right-most mapped base among the first alignments of all genomic records contained in the access unit, irrespective of the strand

3.5
access unit range
genomic range comprised between the access unit start position and the right-most genomic record position among all genomic records contained in the access unit

3.6
access unit covered region
genomic range comprised between the access unit start position and the access unit end position inclusive

3.7
alignment
information describing the similarity between a sequence (typically a sequencing read) and a reference sequence (for instance, a reference genome)

3.8
box
object-oriented building unit defined by a unique type identifier and length

3.9
cluster
aggregation of genomic records

3.10
cluster signature
signature
sequence of nucleotides that is common to most or all genomic records belonging to a cluster

3.11
container box
box (3.8) whose sole purpose is to contain and group a set of related boxes

3.12
data stream
set of *packets* (3.20) transporting the same data type

3.13
extended access unit start position
position of the left-most mapped base among all alignments of all genomic records contained in the access unit, irrespective of the strand

3.14
extended access unit end position
position of the right-most mapped base among all alignments of all genomic records contained in the access unit, irrespective of the strand

3.15
file format
set of data structures for the storage of coded information

3.16
genomic position
position
integer number representing the zero-based position of a nucleotide within a reference sequence

3.17
genomic region
region
genomic interval between a start nucleotide position and an end nucleotide position, inclusive

3.18
genomic range
range
interval of positions on a reference sequence defined by a start position s and an end position e such that $s \le e$; the start and the end positions of a genomic range are always included in the range

3.19
mapped base
base of the aligned read that either matches the corresponding base on the reference sequence or can be turned into the corresponding base on the reference sequence via a substitution

3.20
packet
transmission unit transporting segments of any of the data structures defined in this document

3.21
reference genome
representative example of the sequences for a species' genetic material

Note 1 to entry: Genetic material meaning the sequences of the DNA molecules present in a typical cell of that species.

3.22
reference sequence
nucleic acid sequence with biological relevance

Note 1 to entry: Each reference sequence is indexed by a one-dimensional integer coordinate system whereby each integer within range identifies a single nucleotide. Coordinate values can only be equal to or larger than zero. The coordinate system in the context of this standard is zero-based (i.e. the first nucleotide has coordinate 0 and it is said to be at position 0) and linearly increasing within the string from left to right.

3.23
genomic segment
segment
contiguous sequence of nucleotides, typically output of the sequencing process and sequenced from one strand of a template

3.24
sequence read
read
readout, by a specific technology more or less prone to errors, of a continuous part of a nucleic acid molecule extracted from an organic sample

3.25
syntax field
element of data represented in the data format

3.26
template
genomic sequence that is produced by a sequencing machine as a single unit

Note 1 to entry: A template can be made of one or more segments, being called single-end sequencing read when it only has one segment and paired-end sequencing read when it has two segments.

3.27
transport format
set of data structures for the transport of coded information

3.28
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 Structure of coded genomic data

5.1 Genomic records

The genomic record, in this document, is a data structure consisting of either a single sequence read, or paired sequence reads, and its associated sequencing and alignment information. The genomic record may contain detailed mapping and alignment data, a single or paired read identifier (read name) and quality values.

When alignment information is present, the genomic record position is defined as the position of the left-most mapped base of the genomic record on the reference genome. Genomic record positions are 0-based in the ISO/IEC 23092 series. In case of multiple alignments, the position of the first alignment in the record is considered; in such a case, the first alignment shall be the one with the leftmost position among all the alignments with the best score.

In case of unmapped reads (i.e. no alignment information present) the notion of position does not apply to the genomic record.

In case of aligned content, bases that are present in the reads of the genomic record and not present in the reference sequence (*insertions*) and bases preserved by the alignment process but not mapped on the reference sequence (*soft clips*) do not have mapping positions.

Table 1 enumerates all the types of data that a genomic record can contain. ISO/IEC 23092-2 defines technology that allows coding all and only those types of data into a set of descriptors; data, and consequently descriptors, which are mandatory or optional, are also specified in ISO/IEC 23092-2, as well as how they are used to represent multiple alignments.

Table 1 — Genomic records

Data	Semantics
Record identifier	name of the record (e.g. read names)
Sequence reads	sequencing readout, as one or more strings of bases
Quality values	quality scores of the sequence reads
Strandedness	information about the strandedness of each read of the Record
Length	length of the sequence reads
Position	position on the reference genome of the left-most mapped genomic record base
Pairing	position or distance of the mate reads (e.g. in a pair)
Flags	technical, additional alignment information (duplicates, proper pairs, failures)
Mismatches	information about position and type of each mismatch in mapped records
Clips	information about clipped bases (soft and hard clips) in mapped records
Mapping scores	mapping scores for an alignment
Multiple alignments	information about the number of alignments and the alternative alignment information about each segment of the record
Group	read group the genomic record belongs to

ISO/IEC 23092-2 defines an output record format for all types of data in Table 1. These records shall be generated by decoders compliant to ISO/IEC 23092-2 as output of the decoding process.

5.2 Data classes

Six data classes are specified to classify genomic records according to the result of the mapping of the encoded sequence reads against one or more reference sequences.

In the case of more than one read in a template, if both reads are mapped, the genomic record belongs to the class of the read with the highest class_ID. In case of multiple alignments, the genomic record belongs to the class of the first alignment in the record.

The data classes and their descriptions are specified in Table 2.

Table 2 — Data classes

Class ID	Class name	Record content
1	CLASS_P	Only reads perfectly matching to the reference sequence.
2	CLASS_N	Reads perfectly matching to the reference sequence or containing mismatches which are unknown bases only.
3	CLASS_M	Reads perfectly matching to the reference sequence or containing substitutions or unknown bases, but no insertions, no deletions, no splices and no clipped bases.
4	CLASS_I	Reads perfectly matching to the reference sequence or containing substitutions, unknown bases, insertions, deletions, splices or clipped bases.
5	CLASS_HM	Paired-end reads with only one mapped read.
6	CLASS_U	Unmapped reads only.

Genomic records of each data class are coded by means of several descriptors; conversely, a descriptor is a coding element needed to represent part of the information. Descriptors for each data class are specified in ISO/IEC 23092-2.

Descriptors are coded in blocks. Blocks are defined in subclause 6.5.5. A sequence of block payloads of a single descriptor composes a descriptor stream. All block payloads in a descriptor stream contain compressed descriptors of a single type representing reads of the same data class.

5.3 Access units

Access units (AUs) are data structures containing a coded representation of genomic information and optionally related metadata to facilitate the bitstream access and manipulation. An access unit contains either genomic records belonging to the same data class or a fragment of a reference sequence.

The access unit is the smallest data organization that can be decoded by a decoder compliant with ISO/IEC 23092-2.

Access units are orthogonal to descriptor streams: an access unit is composed of all and only those blocks of the descriptor streams that are necessary to decode the information contained in a cluster of records of a given data class.

An access unit can be of several types according to the class of the coded data.

Table 3 — Access unit type

Access unit type		Class of data
Name	**Value**	
P_TYPE_AU	1	CLASS_P
N_TYPE_AU	2	CLASS_N
M_TYPE_AU	3	CLASS_M
I_TYPE_AU	4	CLASS_I
HM_TYPE_AU	5	CLASS_HM
U_TYPE_AU	6	CLASS_U

Depending on the type of coded information, an access unit can be decoded either independently of any other access unit or using information contained in other access units.

5.4 Datasets

A dataset is a data structure containing headers and access units. The set of access units composing the dataset constitutes the dataset payload.

One or more datasets are assembled into a dataset group.

5.5 Selective access

In the case of selective access to a genomic region comprised between a *start* genomic position and an *end* genomic position the decoder shall return: a) all the access units whose covered region overlaps the region defined by *start* and *end* with at least one base, and the parameter sets that are needed to decode them; b) at least the reference portion that is necessary to decode the access units identified in a).

In the case of selective access to signed content identified by a U_cluster_signature signature the decoder shall return all the access units whose signature corresponds to U_cluster_signature, and the parameter sets that are needed to decode them. Examples of selective access strategies are described in Annex B.

6 Data format

6.1 Format structure

6.1.1 General

Table 4 presents the overall data structures and hierarchical encapsulation levels.

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 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 4 and Figure 5. Mandatory boxes are represented with solid borders. When no entry is present in the *Scope* column, scope is both *File* and *Transport*. See the specification of each individual box for the assumptions when the optional boxes are not present. If the box key is represented in *italic* format in Table 4, the corresponding box is represented either with no Key and no Length, but only Value in the gen_info format, as specified in subclause 6.3, for all boxes but offset, or as specified in subclause 6.6.5.1 for the offset box.

Table 4 — Format structure and encapsulation levels

Box key (with hierarchical level)					Subclause	Scope	Mandatory
flhd					6.5.1		*
dgcn					6.5.2	File	*
	dghd				6.5.2.2		*
	rfgn				6.5.2.3		
	rfmd				6.5.2.4		
	labl				6.5.2.5		
	lbll				6.5.2.5.4		
	dgmd				6.5.2.6		
	dgpr				6.5.2.7		
	dmtl				6.7.3	Transport	*
	dtcn				6.5.3.1	File	*
		dthd			6.5.3.2		*
		mitb			6.6.2.1	File	
		pars			6.5.3.5		*
		dtmd			6.5.3.3		
		dtpr			6.5.3.4		
		dmtb			6.7.4	Transport	*
		dscn			6.6.3.1	File	
			dshd		6.6.3.2	File	
			dspr		6.6.3.3	File	
		aucn			6.5.4.1		*
			auhd		6.5.4.3		*
			auin		6.5.4.4		
			aupr		6.5.4.5		
			block		6.5.5		*
				block_header	6.5.5.2		
	offset	*offset*			6.6.4	File	
packet					6.7.5	Transport	*
	packet_header				6.7.5.2	Transport	*

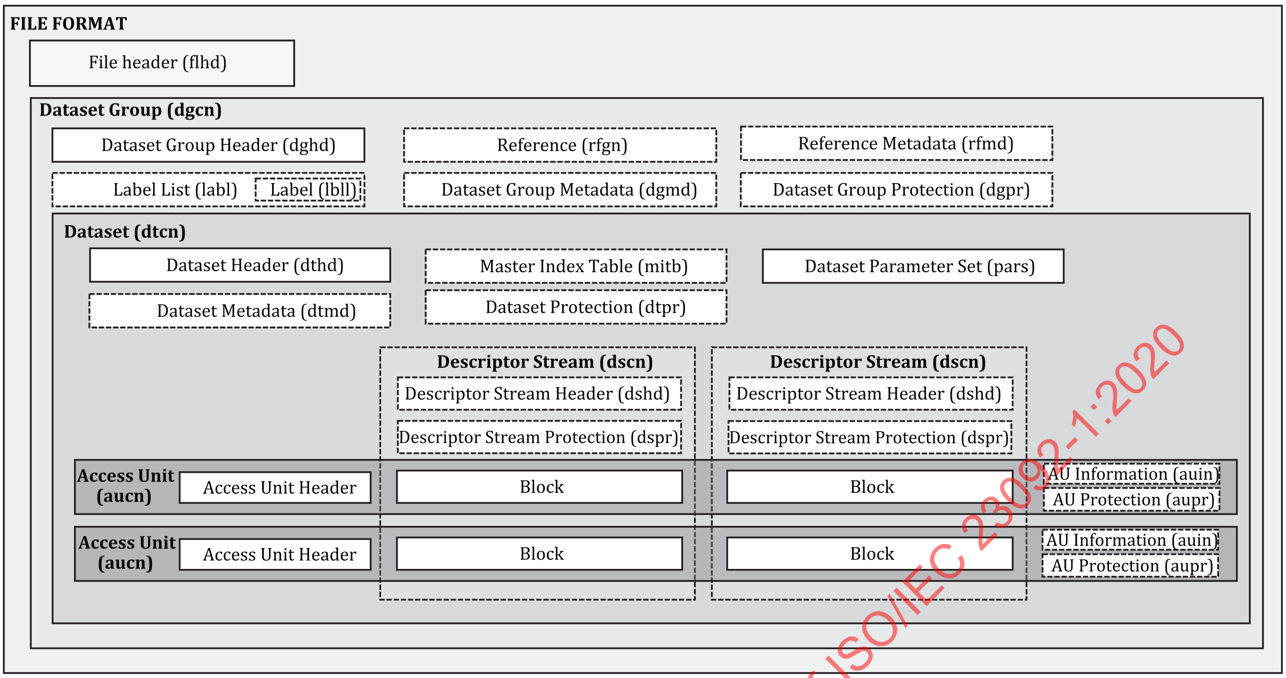

Figure 4 — Data structures hierarchy for storage

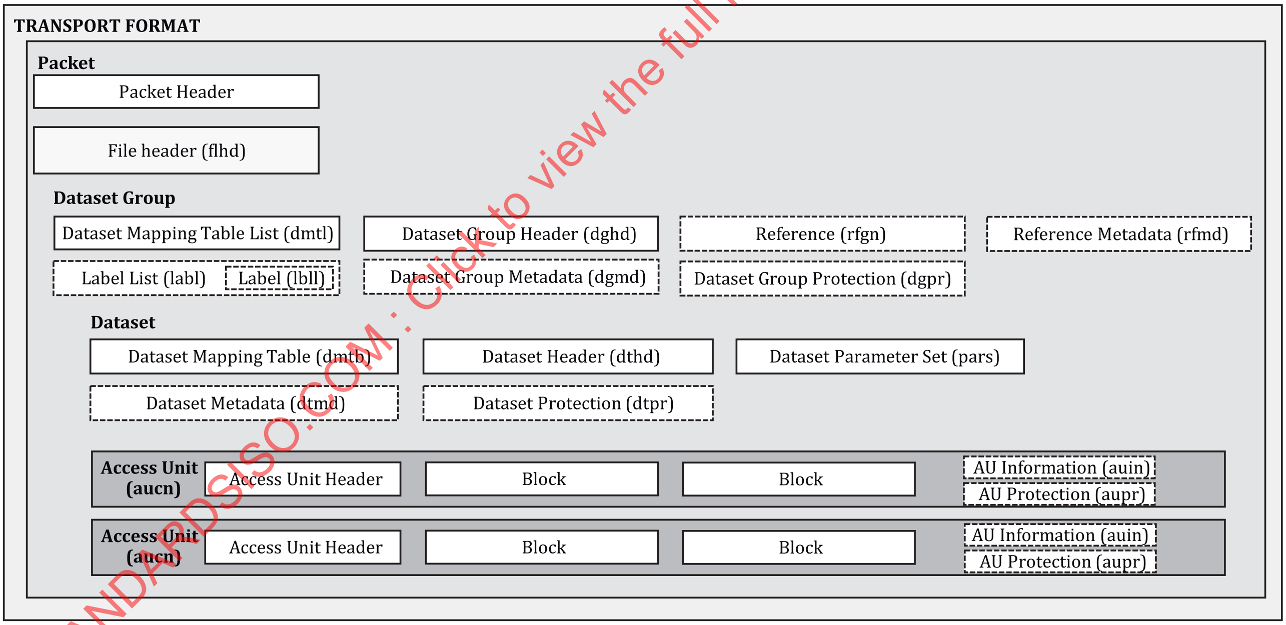

Figure 5 — Data structures hierarchy for transport

In transport format, any box represented in Figure 5 shall be encapsulated in one or more packets, as specified in subclause 6.7.5. The dataset group and dataset are represented in Figure 5 for clarity, but the corresponding container boxes (dgcn and dtcn) 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, access unit and descriptor stream) shall be ordered according to the hierarchy specified in Table 4.

2) The box order inside the containers dgcn, dtcn, dscn, and aucn is specified in Table 9, Table 19, Table 25 and Table 32, 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, 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, dscn, and aucn shall not be broken by any box external to the container box, apart from the offset box, as specified in subclause 6.6.4.

In transport format

1) The dataset_mapping_table_list, dataset_mapping_table and file header boxes shall be decoded first, and then all other boxes according to the hierarchy specified in Table 4.

3) It is recommended to transmit the boxes in hierarchical order, as specified in Table 4.

4) It is recommended, when possible, to interleave boxes with hierarchical level equal to 1 (second column in Table 4) and belonging to different dataset groups, and the dthd boxes belonging to different datasets, before transmitting all other boxes.

6.2 Syntax and semantics

6.2.1 Method of specifying syntax in tabular form

Table 5 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 5 — Constructs used to express the conditions when data elements are present

Construct	Description
`if (condition) { data_element . . . }`	If the condition is true, then the first group of data elements occurs next in the bitstream.
`else { data_element . . . }`	If the condition is not true, then the second group of data elements occurs next in the bitstream.
`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 6.

Table 6 — 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+1th element of an array of data.
`data_element[m][n]`	`data_element[m][n]` is the m+1th,n+1th element of a two-dimensional array of data.
`data_element[l][m][n]`	`data_element[l][m][n]` is the l+1th, m+1th,n+1th 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

byte_aligned( ) is specified as follows:

- If the current position in the bitstream is on a byte boundary, i.e., the next bit in the bitstream is the first bit in a byte, the return value of byte_aligned( ) is equal to TRUE.
- Otherwise, the return value of byte_aligned( ) is equal to FALSE.

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

KLV (Key Length Value) format is used for all the data structures listed in Table 4 but the block, block_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 block, block_header, packet and packet_header data structures have no Key and no Length, but only Value.

The offset data structure is specified in subclause 6.6.4.

All syntax tables specified in subclauses 6.5, 6.6 and 6.7 and related subclauses, 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.

6.4 Output data unit

This subclause specifies the output of the decapsulation processes specified in subclauses 6.5.2.3.6, 6.5.3.5.1 and 6.5.4.

Table 7 — Data unit syntax

Syntax	Type	Remarks
`data_unit() {`		
`data_unit_type`	u(8)	
`if (data_unit_type == 0) {`		
`data_unit_size`	u(64)	
`raw_reference()`		As specified in ISO/IEC 23092-2
`}`		
`else if (data_unit_type == 1) {`		
`reserved`	u(10)	
`data_unit_size`	u(22)	
`parameter_set()`		As specified in ISO/IEC 23092-2
`}`		
`else if (data_unit_type == 2){`		
`reserved`	u(3)	
`data_unit_size`	u(29)	
`access_unit()`		As specified in ISO/IEC 23092-2
`}`		
`else /*(data_unit_type > 2)*/{`		
`/*skip data unit*/`		
`}`		
`}`		

data_unit_type and data_unit_size shall be filled as specified in subclauses 6.5.2.3.6, 6.5.3.5.1 and 6.5.4.

6.5 Data structures common to file format and transport format

6.5.1 File header

6.5.1.1 General

This box is mandatory and provides information about the version of this specification and of the set of other specifications the file or stream complies with.

6.5.1.2 Syntax

Table 8 — File header syntax

Syntax	Key	Type	Remarks
file_header {	*flhd*		
`major_brand`		c(6)	
`minor_version`		c(4)	
`for (i=0;i<num_compatible_brands;i++) {`			
`compatible_brand[i]`		c(4)	
}			
}			

6.5.1.3 Semantics

major_brand is the major brand identifier. The value is equal to the 6-character code “MPEG-G”.

minor_version is an informative set of four characters for the minor version of the major brand of this document 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

num_compatible_brands is inferred from the Length field in the file_header *gen_info* header as follows: num_compatible_brands = (Length – 22) / 4.

compatible_brand[i] is a 4-character code representing a compatible brand.

6.5.2 Dataset group

6.5.2.1 General

The dataset group is a collection of one or more datasets.

The relevant container box (*dgcn* Key in Table 9) is mandatory in file format, forbidden in transport format.

Child boxes may be present or not, according to the column *Mandatory* in Table 4. Child boxes marked with suffix “[]” after their name in the Syntax column of Table 9 may be present in multiple instances.

Table 9 — Dataset group syntax

Syntax	Key	Type	Remarks
dataset_group {	*dgcn*		
dataset_group_header	dghd	gen_info	As specified in subclause 6.5.2.2
reference[]	rfgn	gen_info	As specified in subclause 6.5.2.3
reference_metadata[]	rfmd	gen_info	As specified in subclause 6.5.2.4
label_list	labl	gen_info	As specified in subclause 6.5.2.5
DG_metadata	dgmd	gen_info	As specified in 6.5.2.6
DG_protection	dgpr	gen_info	As specified in 6.5.2.7
for (i=0;i<num_datasets;i++) {			num_datasets: as specified in subclause 6.5.2.2
dataset[i]	dtcn	gen_info	As specified in subclause 6.5.3.1
}			
}			

6.5.2.2 Dataset group header

6.5.2.2.1 General

This is a mandatory box describing the content of a dataset group.

6.5.2.2.2 Syntax

Table 10 — Dataset group header syntax

Syntax	Key	Type	Remarks
dataset_group_header {	*dghd*		
dataset_group_ID		u(8)	
version_number		u(8)	
for (i=0;i<num_datasets;i++) {			
dataset_ID[i]		u(16)	
}			
}			

6.5.2.2.3 Semantics

dataset_group_ID identifies a dataset group. Each value shall be unique among all dataset_group_ID fields in the file or stream.

version_number is the version number of the dataset group. The version number shall be incremented by 1 whenever the definition of the dataset group identified by dataset_group_ID changes. Upon reaching the value 255, it wraps around to 0.

dataset_ID is an integer number identifying the dataset in the dataset group. This field shall not take the same value more than once within the dataset group.

NOTE num_datasets is inferred from the Length field of datasets_group_header *gen_info* header as follows: num_datasets = (Length – 14) / 2.

6.5.2.3 Reference

6.5.2.3.1 General

This is an optional box containing the information needed to retrieve an external or internal reference, and its description as a set of reference sequences.

It may be present in multiple instances in the same dataset group. If so, any instance shall have a different value of reference_ID.

6.5.2.3.2 Syntax

Table 11 — Reference box syntax

Syntax	Key	Type	Remarks
reference {	*rfgn*		
dataset_group_ID		u(8)	
reference_ID		u(8)	
reference_name		st(v)	
reference_major_version		u(16)	
reference_minor_version		u(16)	
reference_patch_version		u(16)	
seq_count		u(16)	
for (seqID=0;seqID<seq_count;seqID++) {			
sequence_name[seqID]		st(v)	
if (minor_version != ‘1900’) {			minor_version as specified in subclause 6.5.2
sequence_length[seqID]		u(32)	
sequence_ID[seqID]		u(16)	
}			
}			
reserved		u(7)	
external_ref_flag		u(1)	
if (external_ref_flag) {			
ref_uri		st(v)	As specified in subclause 6.5.2.3.4
checksum_alg		u(8)	
reference_type		u(8)	
if (reference_type == MPEGG_REF) {			
external_dataset_group_ID		u(8)	
external_dataset_ID		u(16)	
if (minor_version == ‘1900’)			minor_version as specified in subclause 6.5.2
ref_checksum		u(checksum_size)	As specified in 6.5.2.3.7
}			
if (minor_version != ‘1900’ || reference_type != MPEGG_REF) {			minor_version as specified in subclause 6.5.2
for(seqID=0;seqID<seq_count;seqID++) {			
ref_seq_checksum[seqID]		u(checksum_size)	As specified in subclause 6.5.2.3.7

Table 11 *(continued)*

Syntax	Key	Type	Remarks
`}`			
`}`			
`}`			
`else {`			
`internal_dataset_group_ID`		`u(8)`	
`internal_dataset_ID`		`u(16)`	
`}`			
`}`			

6.5.2.3.3 Semantics

dataset_group_ID is the identifier of the dataset group including this box. It shall have the same value as the dataset_group_ID field in the dataset group header of the same dataset group, as specified in subclause 6.5.2.2.

reference_ID is the identification number of the reference within the dataset group.

reference_name is a string representing a human readable name of the reference.

reference_major_version is the reference major version.

reference_minor_version is the reference minor version.

reference_patch_version is the reference patch version.

seq_count is the number of reference sequences contained in the reference genome.

sequence_name is an unambiguous string identifier for each reference sequence contained in the reference.

sequence_length is the number of nucleotides of each reference sequence contained in the reference.

sequence_ID is reference sequence integer identifier.

external_ref_flag is a flag specifying whether the reference is either another dataset of the same bitstream, as specified in subclause 6.5.3, with dataset_type equal to 2 (external_ref_flag equal to 0), or a reference external to the bitstream (external_ref_flag equal to 1).

ref_uri as specified in subclause 6.5.2.3.4.

reference_type specifies the type of the external reference and can take any of the values in the first column of Table 12.

Table 12 — reference_type values

Value	Name	Semantics
0	MPEGG_REF	Reference encoded as a dataset, as specified in subclause 6.5.3, identified by fields external_dataset_group_ID and external_dataset_ID when external_ref_flag is equal to 1, or by fields internal_dataset_group_ID and internal_dataset_ID, when external_ref_flag is equal to 0, in a bitstream compliant to this document. The dataset shall have dataset_type, as specified in subclause 6.5.3.2, equal to 2.
1	RAW_REF	Raw reference, as specified in ISO/IEC 23092-2.
2	FASTA_REF	Reference of type FASTA, as specified in subclause 6.5.2.3.5
3 to 0xFF		Reserved for future use

external_dataset_group_ID is the identifier of the dataset group containing the external reference, in case ref_uri points to a reference coded in compliance to the ISO/IEC 23092 series.

external_dataset_ID is the identifier of the dataset containing the external reference, in case ref_uri points to a reference coded in compliance to the ISO/IEC 23092 series. The value shall be equal to one of the dataset_ID belonging to the dataset group identified by external_dataset_group_ID.

ref_checksum is the checksum computed, according to one of the methods specified in subclause 6.5.2.3.7, on the payload of the dataset of type 2 (Value field of dtcn box), as specified in subclause 6.5.3, retrieved using ref_uri, external_dataset_group_ID and external_dataset_ID.

ref_seq_checksum is the checksum computed, according to one of the methods specified in subclause 6.5.2.3.7, for each reference sequence contained in the reference genome, on the content of the corresponding ref_sequence field of the raw reference, as specified in ISO/IEC 23092-2, either as is, if reference_type is equal to RAW_REF, or obtained via the conversion process specified in subclause 6.5.2.3.6, if reference_type is equal to FASTA_REF or MPEGG_REF.

internal_dataset_group_ID is an integer number identifying the dataset group containing the internal reference. An internal reference shall be of type MPEGG_REF, as specified in Table 12.

internal_dataset_ID is an integer number identifying the dataset containing the internal reference. An internal reference shall be of type MPEGG_REF, as specified in Table 12.

6.5.2.3.4 ref_uri semantics

ref_uri shall be compliant with IETF RFC 3986 and IETF RFC 7320.

The IETF RFC 3986 specification is partially summarized in Annex A.

6.5.2.3.5 Supported FASTA format

The FASTA format[2] supported by this document is represented as a series of lines in ASCII text format.

The first line in the FASTA shall start with a ">" (greater-than) symbol.

Ignore lines that only contain non printable characters and any comment line starting with a semi-colon.

Each sequence of characters following a ">" (greater-than) symbol and up to the first whitespace character shall be interpreted as the identifier (a.k.a. name) of the sequence of nucleotides represented by the following one or more lines.

Each sequence of characters following a ">" (greater-than) symbol and up to an end of line character shall be followed by one or more lines of symbols representing nucleotides.

The following is an example of supported FASTA.

Line	Content	Description
1	>1 dna:chromosome chromosome:GRCh37:1:1:249250621:1	1 = first sequence identifier
2	ACGTTGACTATCGATCTATTAGCGGCGATGCA	Sub-sequences of nucleotides representing the entire first sequence
3	TGACTATCGATCTATTAGCGGCGATGCTTCCA	
4	ACGTTGACAAACCGATAAGCGGCGATGCAAAC	
…	…	
N	>2 dna:chromosome chromosome:GRCh37:2:1:243199373:1	2 = second sequence identifier
N+1	TGACTATCGATCTATTAGCGGCGATGCTTCCA	Sub-sequences of nucleotides representing the entire second sequence
N+2	ACGTTGACAAACCGATAAGCGGCGATGCAAAC	
N+3	TTGACAAACCGATAAGCGGCGATGCAAACAGT	
…	…	

Line	Content	Description
...	...	...

A compliant codec will ignore all new line characters and any comment line starting with a semi-colon.

6.5.2.3.6 Conversion to raw reference

The reference either pointed by ref_uri (when external_ref_flag is equal to 1) and, if reference_type is equal to MPEGG_REF, identified by external_dataset_group_ID and external_dataset_ID, or (when external_ref_flag is equal to 0) identified by internal_dataset_group_ID and internal_dataset_ID fields, shall be converted into a raw reference structure, as specified in ISO/IEC 23092-2, according to the process described below.

— If either external_ref_flag is equal to 0, or external_ref_flag is equal to 1 and reference_type is equal to MPEGG_REF, as specified in Table 12, the corresponding dataset shall be decapsulated, according to subclause 6.5.3, and the output data units shall be decoded, according to the decoding process specified in ISO/IEC 23092-2.

— Else, if external_ref_flag is equal to 1 and reference_type is equal to RAW_REF, as specified in Table 12, no decapsulation is needed.

— Else, if external_ref_flag is equal to 1 and reference_type is equal to FASTA_REF, as specified in Table 12, the FASTA reference, as specified in subclause 6.5.2.3.5, shall be converted into a raw reference, by filling each ref_sequence field with the corresponding base characters in the FASTA reference, after conversion to uppercase letters and ignoring new line characters; seq_start field shall be set to 0, seq_end field shall be set to the number of characters composing each sequence minus 1, and the sequence_ID field structure shall be set either incrementally, starting from 0, for any reference sequence present in the FASTA reference, if minor_version is equal to '1900' or to the value of sequence_ID field in the reference box, as specified in subclause 6.5.2.3.2.

In all of the above three cases the output raw reference shall be encapsulated as payload of a data unit, as specified in subclause 6.4, with:

— data_unit_type equal to 0,

— data_unit_size equal to the sum of 9 (the number of bytes used for data_unit_type and data_unit_size) and the number of bytes composing the raw_reference structure.

6.5.2.3.7 Checksum

The identification of the hash function to be used to verify the identity of the related reference (ref_checksum field) or reference sequences (ref_seq_checksum[i] fields) is performed using checksum_alg, as specified in subclause 6.5.2.3. Two values of checksum_alg are defined in Table 13, while other values are reserved for future use.

Table 13 — Checksum values

checksum_alg value	Checksum algorithm	checksum_size	Rationale
0x00	MD5	128	Supported as checksum algorithm only for backward compatibility, but it is not recommended for the creation of new content due to the extensive collision vulnerabilities it suffers.
0x01	SHA-256	256	Currently recommended for all hash function-based applications and it shall be used for the integrity check of all new content.
0x02 to 0xFF			Reserved for future use.

6.5.2.4 Reference metadata

6.5.2.4.1 General

This is an optional box containing metadata associated to a reference.

6.5.2.4.2 Syntax

Table 14 — Reference Metadata syntax

Syntax	Key	Type	Remarks
reference_metadata {	*rfmd*		
dataset_group_ID		u(8)	
reference_ID		u(8)	
reference_metadata_value()			As specified in ISO/IEC 23092-3
}			

6.5.2.4.3 Semantics

dataset_group_ID is an integer number identifying the dataset group including this reference_metadata.

reference_ID is a unique identification number of the reference to which this reference_metadata refers to. It shall be equal to the reference_id value of one of the reference boxes, as specified in subclause 6.5.2.3, present in the dataset group.

reference_metadata_value() contains reference related metadata, as specified in ISO/IEC 23092-3.

6.5.2.5 Label List

6.5.2.5.1 General

This box lists the labels, as specified in subclause 6.5.2.5.4, associated to a dataset group.

6.5.2.5.2 Syntax

Table 15 — Label list syntax

Syntax	Key	Type	Remarks
label_list {	*labl*		
dataset_group_ID		u(8)	
num_labels		u(16)	
for (h=0; h<num_labels; h++) {			
label[h]	*lbll*	gen_info	As specified in subclause 6.5.2.5.4
}			
}			

6.5.2.5.3 Semantics

dataset_group_ID is the identifier of the dataset group including this label list. It shall have the same value as the dataset_group_ID field in the dataset group header of the same dataset group, as specified in subclause 6.5.2.2.

num_labels is the total number of labels in the label list.

6.5.2.5.4 Label

6.5.2.5.4.1 General

A label is an identifier associated to one or more datasets, genomic regions and/or classes.

6.5.2.5.4.2 Syntax

Table 16 — Label syntax

Syntax	Key	Type	Remarks
label {	*lbll*		
label_ID		st(v)	
num_datasets		u(16)	
for (i=0;i<num_datasets;i++) {			
dataset_ID[i]		u(16)	
num_regions[i]		u(8)	
for (j=0;j<num_regions[i];j++) {			
seq_ID[i][j]		u(16)	
num_classes[i][j]		u(4)	
for (k=0;k<num_classes[i][j];k++) {			
class_ID[i][j][k]		u(4)	
}			
start_pos[i][j]		u(40)	
end_pos[i][j]		u(40)	
}			
}			
while(!byte_aligned( ))			As specified in subclause 6.2
nesting_zero_bit		f(1)	Equal to 0
}			

6.5.2.5.4.3 Semantics

label_ID is a string representing the label identifier in the label list specified in subclause 6.5.2.5. The variable stringLength for this field, as specified in subclause 6.2.3, concerning the st(v) data type, shall be higher than 0.

num_datasets is the number of datasets containing regions labelled by label_ID.

dataset_ID is the identifier of a dataset labelled by label_ID. It shall take one of the values of dataset_ID listed in the dataset group header of the same dataset group, as specified in subclause 6.5.2.2.

num_regions: is the number of regions labelled by label_ID in the dataset.

seq_ID is the sequence identifier. It shall take the value of one of the sequence_ID fields of at least one of the reference boxes included in the dataset group, as specified in subclause 6.5.2.3.

num_classes is the number of classes labelled by label_ID in the region.

class_ID identifies the data class in the region labelled by label_ID, as specified in Table 2.

start_pos is the position of the left-most mapped base among the first alignments of all genomic records encoded in the access units covering the region, irrespective of the strand.

end_pos is the position of the right-most mapped base among the first alignments of all genomic records encoded in the access units covering the region, irrespective of the strand.

6.5.2.6 Dataset group metadata

6.5.2.6.1 General

This is an optional box containing metadata associated to a dataset group.

6.5.2.6.2 Syntax

Table 17 — Dataset group metadata syntax

Syntax	Key	Type	Remarks
DG_metadata {	*dgmd*		
if (minor_version != '1900') {			minor_version as specified in subclause 6.5.1
dataset_group_ID		u(8)	
}			
DG_metadata_value()			As specified in ISO/IEC 23092-3
}			

6.5.2.6.3 Semantics

DG_metadata_value() contains the dataset group metadata, specified in ISO/IEC 23092-3.

6.5.2.7 Dataset group protection

6.5.2.7.1 General

This is an optional box containing protection information associated to a dataset group.

When present this box contains information that a decoder needs to properly handle a protected dataset group.

6.5.2.7.2 Syntax

Table 18 — Dataset group protection syntax

Syntax	Key	Type	Remarks
DG_protection {	*dgpr*		
if (minor_version != '1900') {			minor_version as specified in subclause 6.5.1
dataset_group_ID		u(8)	
}			
DG_protection_value()			As specified in ISO/IEC 23092-3
}			

6.5.2.7.3 Semantics

DG_protection_value() contains the dataset group protection information, specified in ISO/IEC 23092-3.

6.5.3 Dataset

6.5.3.1 General

A dataset is a collection of access units encoding either records or a reference.

The corresponding container box (*dtcn* in Table 19) is mandatory in file format, forbidden in transport format.

Child boxes may be present or not, according to the column "Mandatory" in Table 4. Child boxes marked with suffix "[]" after their name in the Syntax column of Table 19 may be present in multiple instances.

Table 19 — Dataset syntax

Syntax	Key	Type	Remarks
dataset {	*dtcn*		
dataset_header	dthd	gen_info	As specified in subclause 6.5.3.2
DT_metadata	dtmd	gen_info	As specified in 6.5.3.3
DT_protection	dtpr	gen_info	As specified in 6.5.3.4
dataset_parameter_set[]	pars	gen_info	As specified in subclause 6.5.3.3
if (MIT_flag == 1) {			As specified in subclause 6.5.3.2
master_index_table	mitb	gen_info	As specified in subclause 6.6.2.1
}			
access_unit[]	aucn	gen_info	As specified in subclause 6.5.4
if (block_header_flag == 0) {			
descriptor_stream[]	dscn	gen_info	As specified in subclause 6.6.3
}			
}			

6.5.3.2 Dataset header

6.5.3.2.1 General

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

6.5.3.2.2 Syntax

Table 20 — Dataset header syntax

Syntax	Key	Type	Remarks
dataset_header {	*dthd*		
dataset_group_ID		u(8)	
dataset_ID		u(16)	
version		c(4)	
multiple_alignment_flag		u(1)	
byte_offset_size_flag		u(1)	
non_overlapping_AU_range_flag		u(1)	
pos_40_bits_flag		u(1)	

Table 20 *(continued)*

Syntax	Key	Type	Remarks
block_header_flag		u(1)	
if (block_header_flag) {			
MIT_flag		u(1)	
CC_mode_flag		u(1)	
}			
else {			
ordered_blocks_flag		u(1)	
}			
seq_count		u(16)	
if (seq_count > 0) {			
reference_ID		u(8)	
for (seq=0;seq<seq_count;seq++) {			
seq_ID[seq]		u(16)	
}			
for (seq=0;seq<seq_count;seq++) {			
seq_blocks[seq]		u(32)	
}			
}			
dataset_type		u(4)	
if (MIT_flag == 1) {			
num_classes		u(4)	
for (ci=0;ci<num_classes;ci++) {			
clid[ci]		u(4)	
if (!block_header_flag) {			
num_descriptors[ci]		u(5)	
for(di=0;di<num_descriptors[ci];di++) {			
descriptor_ID[ci][di]		u(7)	
}			
}			
}			
}			
parameters_update_flag		u(1)	
alphabet_ID		u(7)	
num_U_access_units		u(32)	
if (num_U_access_units > 0) {			
reserved		u(62)	For backward compatibility
U_signature_flag		u(1)	
if(U_signature_flag) {			
U_signature_constant_length		u(1)	
if (U_signature_constant_length){			
U_signature_length		u(8)	
}			
}			
reserved_flag		u(1)	
if (reserved_flag){			
reserved		u(8)	For backward compatibility
}			

Table 20 *(continued)*

Syntax	Key	Type	Remarks
reserved_flag		u(1)	
}			
if (seq_count > 0) {			
tflag[0]		f(1)	Equal to 1
thres[0]		u(31)	
for (i=1;i<seq_count;i++) {			
tflag[i]		u(1)	
if(tflag[i] == 1)			
thres[i]		u(31)	
else /* tflag[i] == 0 */			
/* thres[i] = thres[i-1] */			
}			
}			
while(!byte_aligned( ))			As specified in subclause 6.2
nesting_zero_bit		f(1)	Equal to 0
}			

6.5.3.2.3 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

multiple_alignment_flag: if set to 1 it indicates the presence of multiple alignments in the dataset. It shall never be set to 1 if dataset_type is equal to 2.

byte_offset_size_flag: if equal to 0, the variable byteOffsetSize used in the master index table, as specified in subclause 6.6.2.1, and representing the number of bits used to encode the master index table fields named AU_byte_offset and block_byte_offset, is equal to 32; if set to 1, the variable byteOffsetSize is equal to 64.

non_overlapping_AU_range: if set to 1, all access units with the same AU_type in the dataset have non-overlapping ranges.

pos_40_bits_flag is set to 1 when the mapping positions are expressed as 40 bits integers. Otherwise all mapping positions are expressed as 32 bits integers. In the scope of this document, the value of the variable posSize is set to 32 when pos_40_bits is equal to 0 and set to 40 otherwise.

block_header_flag: if set, all blocks composing the dataset are preceded by a block header, as specified in subclause 6.5.5.2. It is always set to 1 in transport format. See also subclause 6.5.3.2.5.

MIT_flag: if set, the master index table, as specified in subclause 6.6.2.1 is present in the dataset. Otherwise, the master index table is not present in the dataset. It is always equal to 0 in transport format and set to 1 by default when block_header_flag is equal to 0.

CC_mode_flag: if set, two access units of the same type, as specified in Table 3, cannot be separated by access units of a different type in the storage device. If equal to 0, access units are ordered by access unit start position in the storage device. See also subclause 6.5.3.2.5.

ordered_blocks_flag: if set, blocks are ordered in the descriptor stream by increasing value of the entry AU_start_position of the master index table, as specified in subclause 6.6.2.1. See also subclause 6.5.3.2.5.

seq_count is the number of reference sequences used in this dataset, in case of file format; in case of transport format and if dataset type is different than 0, it shall be equal to the seq_count field in the reference box, as specified in subclause 6.5.2.3.1, referenced by this dataset by the reference_id field. It shall always be equal to 0 when dataset_type is equal to 0.

reference_ID is a unique identification number of the reference used by the dataset for alignment. It shall take the value of the reference_ID field of any of the reference boxes included in the dataset group including this dataset, as specified in subclause 6.5.2.3. If dataset_type is equal to 2, and if the external_ref_flag field of the reference box pointed by reference_id, as specified in subclause 6.5.2.3, is equal to 0, then the reference box pointed by reference_id, as specified in subclause 6.5.2.3, shall have either a value of the internal_dataset_ID field different than the value of the dataset_ID field in this dataset header, or a value of the internal_dataset_group_ID field different than the value of the dataset_group_ID field in this dataset header.

seq_ID: its value shall correspond to any of the values of the sequence_ID variable in the reference box identified by reference_ID, as specified in subclause 6.5.2.3.

seq_blocks is the number of access units per reference sequence with a different value of access_unit_ID, as specified in subclause 6.5.4. A value of 0 means "unspecified" (e.g., in transport format).

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

num_classes is the number of classes encoded in the dataset.

clid identifies the class of data carried by the access unit, as specified in Table 2. It shall take any of the values defined as Class ID in Table 2. clid[ci+1] shall be greater than clid[ci], for ci in the range between 0 and (num_classes – 2), inclusive. Variable ci is used as a local identifier for the class in the other syntax tables included in the same dataset.

num_descriptors is the maximum number of descriptors per class encoded in the dataset.

descriptor_ID is a descriptor identifier as specified in ISO/IEC 23092-2.

parameters_update_flag: if equal to 1, the fields multiple_alignment_flag, pos_40_bits_flag, U_signature_flag, U_signature_constant_length and U_signature_length in this dataset header are also present in all the dataset parameter sets, as specified in subclause 6.5.3.5.3, contained in the same dataset. In such a case (parameters_update_flag equal to 1), such fields shall have the following values in the dataset header:

- multiple_alignment_flag shall be equal to 1 if the multiple_alignment_flag field in at least one of the dataset parameter sets, as specified in subclause 6.5.3.5, is equal to 1; otherwise it shall be equal to 0.
- pos_40_bits shall be equal to 1 if the pos_40_bits field in at least one of the dataset parameter sets, as specified in subclause 6.5.3.5, is equal to 1; otherwise it shall be equal to 0.
- alphabet_id shall be equal to 1 if the alphabet_id field in at least one of the dataset parameter sets, as specified in subclause 6.5.3.5, is equal to 1; otherwise it shall be equal to 0.
- U_signature_flag shall be equal to 1 if the U_signature_flag field in at least one of the dataset parameter sets, as specified in subclause 6.5.3.5, is equal to 1; otherwise it shall be equal to 0.
- U_signature_constant_length shall be equal to 0.

When parameters_update_flag is equal to 1:

- when used in the access unit header, as specified in subclause 6.5.4.3, such fields will take the value they have in the dataset parameter set, as specified in subclause 6.5.3.5, referenced by the access unit via the parameter_set_ID field, as specified in subclause 6.5.4.3
- when used in the master index table, as specified in subclause 6.6.2.1, such fields will take the values present in the dataset header, as specified in this subclause.

alphabet_ID is the identifier of the alphabet used to encode the cluster signatures. Values are described in Table 21 in subclause 6.5.3.2.4.

num_U_access_units is the total number of access units in the dataset containing encoded data of class U, in case of file format; in case of transport format, it shall always be set to 1 and does not have any specific semantics apart from the use as a flag in Table 20.

U_signature_flag: if set to 1 it indicates the presence of cluster signatures in the access units contained in the dataset.

U_signature_constant_length: if set to 1 all cluster signatures in the dataset have the same length.

U_signature_length is the length of all cluster signatures in the dataset, as number of nucleotides.

tflag: if set to 1 it indicates that it is followed by a threshold **thres.**

thres is a threshold indicating the maximum difference between the access unit covered region and the access unit range.

6.5.3.2.4 Alphabets

Each Alphabet is identified by an alphabet_ID as shown in Table 21:

Table 21 — alphabet_ID semantics

alphabet_ID	$S_{alphabet\_ID}$	bits_per_symbol
0	S_0 = [A, C, G, T, N]	3
1	S_1 = [A, C, G, T, R, Y, S, W, K, M, B, D, H, V, N, -]	5
2 .. 255	reserved	

The notation $S_{alphabet\_ID}$[index] specifies a conversion from a numerical index to an ASCII character corresponding to a symbol of the alphabet identified by alphabet_ID, as specified below.

S_0[0]=’A’, S_0[1]=’C’, S_0[2]=’G’, S_0[3]=’T’ , S_0[4]=’N’

S_1[0]=’A’, S_1[1]=’C’, S_1[2]=’G’, S_1[3]=’T’, S_1[4]=’R’, S_1[5]=’Y’, S_1[6]=’S’, S_1[7]=’W’, S_1[8]=’K’, S_1[9]=’M’, S_1[10]=’B’ , S_1[11]=’D’, S_1[12]=’H’, S_1[13]=’V’, S_1[14]=’N’, S_1[15]=’-’

6.5.3.2.5 Block contiguity

The field block_header_flag is used to enable two possible modes of block contiguity in the file:

- descriptor stream contiguity (DSC) mode: blocks, as specified in subclause 6.5.5, belonging to the same descriptor stream, as specified in subclause 6.6.3, are stored in contiguous areas of the storage device. This mode is enabled by the condition block_header_flag equal to 0.
- access unit contiguity (AUC) mode: blocks, as specified in subclause 6.5.5, belonging to the same access unit, as specified in subclause 6.5.4, are stored in contiguous areas of the storage device. This mode is enabled by the condition block_header_flag equal to 1.

When block_header_flag is equal to 1, the CC_mode_flag field is used to enable two possible modes of access units contiguity in the file, named:

- genomic region contiguity (AUC-GRC) mode: access units are ordered by access unit start position in the storage device. This mode is enabled by the condition CC_mode_flag equal to 0.
- class contiguity (AUC-CC) mode: two access units of one class cannot be separated by access units of a different class in the storage device. This mode is enabled by the condition CC_mode_flag equal to 1.

No other block contiguity modes are allowed by this document.

When block_header_flag is equal to 0 (DSC mode), the field ordered_blocks_flag is used to indicate whether the blocks are ordered in the storage device according to the left-most aligned position of the left-most read in the access unit (field AU_ref_start_position in access unit header, as specified in subclause 6.5.4.3, or master index table, as specified in subclause 6.6.2.1). If ordered_blocks_flag is equal to 1, the file offsets for a given descriptor stream and for each block are sorted in ascending order (disregarding blocks for which the block_byte_offset in the master index table is equal to ((1 << byteOffsetSize) -1). In this mode the first byte not belonging to the block is the first byte of the next available block if any (otherwise the descriptor_stream_size, which can be inferred from the Length field of the gen_info header of descriptor stream container box with Key *dscn*, should be used).

If ordered_blocks_flag is equal to 0, the blocks may be stored in any order in the descriptor stream. In order to infer the offset of the first byte not belonging to the block, the decoder has to search, among all offsets provided for the descriptor stream which are not equal to ((1 << byteOffsetSize) -1), the smallest value greater than the offset of the block, if any (otherwise the descriptor_stream_size should be used as above).

6.5.3.3 Dataset metadata

6.5.3.3.1 General

This is an optional box containing metadata associated to the dataset.

6.5.3.3.2 Syntax

Table 22 — Dataset metadata syntax

Syntax	Key	Type	Remarks
DT_metadata {	*dtmd*		
if (minor_version != '1900') {			minor_version as specified in subclause 6.5.1
dataset_group_ID		u(8)	
dataset_ID		u(16)	
}			
DT_metadata_value()			As specified in ISO/IEC 23092-3
}			

6.5.3.3.3 Semantics

DT_metadata_value() contains dataset metadata, as specified in ISO/IEC 23092-3.

6.5.3.4 Dataset protection

6.5.3.4.1 General

This is an optional box containing protection information associated to the dataset.

When present this box contains information that a decoder needs to properly handle a protected dataset.

6.5.3.4.2 Syntax

Table 23 — Dataset protection syntax

Syntax	Key	Type	Remarks
DT_protection {	*dtpr*		
if (minor_version != '1900') {			minor_version as specified in subclause 6.5.1
dataset_group_ID		u(8)	
dataset_ID		u(16)	
}			
DT_protection_value()			As specified in ISO/IEC 23092-3
}			

6.5.3.4.3 Semantics

DT_protection_value() contains dataset protection information, as specified in ISO/IEC 23092-3.

6.5.3.5 Dataset parameter set

6.5.3.5.1 General

This is a mandatory box describing any of the parameter sets associated to the dataset identified by dataset_ID in the dataset group identified by dataset_group_ID.

It may be present in multiple instances in the same dataset.

The decapsulation of this box shall result in a data unit, as specified in subclause 6.4, with:

- data_unit_type equal to 1,
- data_unit_size equal to the sum of 5 (the number of bytes used for data_unit_type and data_unit_size), 2 (the number of bytes for parent_parameter_set_ID and parameter_set_ID, as specified in subclause 6.5.3.5.2) and the number of bytes composing the encoding_parameters() structure, as specified in subclause 6.5.3.5.2, and
- as payload a parameter_set() structure composed of the parent_parameter_set_ID, parameter_set_ID and encoding_parameters() fields, as specified in subclause 6.5.3.5.2.

Such data unit can be dispatched to a decoder compliant with ISO/IEC 23092-2.

6.5.3.5.2 Syntax

Table 24 — Dataset parameter set syntax

Syntax	Key	Type	Remarks
dataset_parameter_set {	*pars*		
dataset_group_ID		u(8)	
dataset_ID		u(16)	
parameter_set_ID		u(8)	
parent_parameter_set_ID		u(8)	
if (minor_version != '1900' && parameters_update_flag) {			As specified in subclause 6.5.3.2
multiple_alignment_flag		u(1)	
pos_40_bits_flag		u(1)	
alphabet_ID		u(8)	
if (num_U_access_units > 0) {			As specified in subclause 6.5.3.2
U_signature_flag		u(1)	
if(U_signature_flag) {			
U_signature_constant_length		u(1)	
if (U_signature_constant_length){			
U_signature_length		u(8)	
}			
}			
}			
while(!byte_aligned( ))			As specified in subclause 6.2
nesting_zero_bit	f(1)		
}			
encoding_parameters()			As specified in ISO/IEC 23092-2
}			

6.5.3.5.3 Semantics

dataset_group_ID is the identifier of the dataset group containing the dataset including this dataset parameter set. It shall be equal to the dataset_group_ID of the containing dataset group.

dataset_ID is the identifier of the dataset including this dataset parameter set. It shall be equal to the dataset_id of the containing dataset.

parameter_set_ID is the identifier of this dataset parameter set within the dataset.

parent_parameter_set_ID is the identifier of any of the dataset parameter sets within the dataset. Referencing an existing dataset parameter set from another parameter set enables the generation of a hierarchy of dataset parameter sets to be associated to an access unit, as specified in subclause 6.5.4. If the value of parent_parameter_set_ID is equal to the value of parameter_set_ID, then the dataset parameter set is at the top level in the hierarchy.

multiple_alignment_flag: if set to 1 it indicates the presence of multiple alignments in the access units associated to this dataset parameter set. It shall never be set to 1 if dataset_type is equal to 2.

pos_40_bits_flag is set to 1 when the mapping positions in the access units associated to this dataset parameter set are expressed as 40 bits integers. Otherwise all mapping positions are expressed as 32 bits integers. In the scope of this document, the value of the variable posSize is set to 32 when pos_40_bits is equal to 0 and set to 40 otherwise.

alphabet_ID is the identifier of the alphabet used to encode the cluster signatures in the access units associated to this dataset parameter set. Values are described in Table 21 in subclause 6.5.3.2.4.

U_signature_flag: if set to 1 it indicates the presence of cluster signatures in the access units associated to this dataset parameter set.

U_signature_constant_length: if set to 1 all cluster signatures in the access units associated to this dataset parameter set have the same length.

U_signature_length is the length of all cluster signatures in the access units associated to this dataset parameter set, as number of nucleotides.

encoding_parameters() is an encoding_parameters() structure as specified in ISO/IEC 23092-2.

6.5.4 Access unit

6.5.4.1 General

The access unit is a collection of one or more blocks representing genomic information.

The decapsulation of this mandatory box shall result in a data unit, as specified in subclause 6.4, with:

- data_unit_type equal to 2,
- data_unit_size equal to the sum of 5 (the number of bytes used for data_unit_type and data_unit_size) and the number of bytes composing the output access unit structure, as specified in subclause 6.5.4.2,
- as payload the output access unit structure, as specified in subclause 6.5.4.2.

Such data unit can be dispatched to a decoder compliant with ISO/IEC 23092-2, along with all the parameter sets that are needed to decode it.

Table 25 — access unit syntax

Syntax	Key	Type	Remarks
access_unit {	*aucn*		
access_unit_header	auhd	gen_info	As specified in subclause 6.5.4.3
AU_information	auin	gen_info	As specified in subclause 6.5.4.4
AU_protection	aupr	gen_info	As specified in subclause 6.5.4.5
if (block_header_flag) {			As specified in subclause 6.5.3.2
for (i=0;i<num_blocks;i++) {			As specified in subclause 6.5.4.3
block[i]			As specified in subclause 6.5.5
}			
}			
}			

6.5.4.2 Access unit decapsulation

Output to this process is an output access unit structure composed of:

- an access_unit_header structure, as specified in subclause 6.5.4.3, where MIT_flag shall be set to 0, and which is composed of:
  - the entire Value field of the access_unit_header child box present in the access unit box, as specified in Table 25, possibly followed, if MIT_flag in the dataset header, as specified in subclause 6.5.3.2, is equal to 1, by
  - the set of fields in the access unit header, as specified in subclause 6.5.4.2, which are enclosed within the if (MIT_flag==0) condition branch (such as sequence_ID, AU_start_position, AU_end_position, etc.) and which shall be derived from the corresponding parameters in the master index table, as specified in subclause 6.6.2.1:
    - The seq index shall be used to infer the sequence_ID field value from the seq_ID array in the dataset header, as specified in subclause 6.5.3.2
    - AU_start_position shall be set to the corresponding AU_start_position field in the master index table, as specified in subclause 6.6.2.1
    - AU_end_position , shall be set to the corresponding AU_end_position field in the master index table, as specified in subclause 6.6.2.1;
- the set of blocks either contained in the access unit box, as specified in Table 25, if block_header_flag is equal to 1, or to be retrieved from the descriptor streams, as specified in subclauses 6.6.2 and 6.6.3, if block_header_flag is equal to 0; in the second case, a block header, as specified in subclause 6.5.5.2, shall be prepended to all the blocks, where the descriptor_ID field shall be equal to the descriptor_ID field of the corresponding descriptor stream header, as specified in subclause 6.6.3.2, and the block_payload_size field shall be equal to the number of bytes composing the block payload as retrieved from the descriptor stream.

6.5.4.3 Access unit header

6.5.4.3.1 General

This mandatory box contains information associated to the access unit.

6.5.4.3.2 Syntax

Table 26 — Access unit header syntax

Syntax	Key	Type	Remarks
access_unit_header {	*auhd*		
`access_unit_ID`		u(32)	
`num_blocks`		u(8)	
`parameter_set_ID`		u(8)	
`AU_type`		u(4)	
`reads_count`		u(32)	
`if (AU_type == N_TYPE_AU ||` `AU_type == M_TYPE_AU ) {`			
`mm_threshold`		u(16)	
`mm_count`		u(32)	
`}`			
`if (dataset_type == 2) {`			
`ref_sequence_id`		u(16)	

Table 26 *(continued)*

Syntax	Key	Type	Remarks
ref_start_position		u(posSize)	posSize as specified in subclause 6.5.3.2.3 or 6.5.3.5.3, depending on parameters_update_flag field
ref_end_position		u(posSize)	posSize as specified in subclause 6.5.3.2.3 or 6.5.3.5.3, depending on parameters_update_flag field
}			
if (MIT_flag == 0) {			As specified in subclause 6.5.3.2
if (AU_type != U_TYPE_AU)			
{			
sequence_ID		u(16)	
AU_start_position		u(posSize)	posSize as specified in subclause 6.5.3.2.3 or 6.5.3.5.3, depending on parameters_update_flag field
AU_end_position		u(posSize)	posSize as specified in subclause 6.5.3.2.3 or 6.5.3.5.3, depending on parameters_update_flag field
if (multiple_alignment_flag) {			As specified in subclause 6.5.3.2 or 6.5.3.5, depending on parameters_update_flag field
extended_AU_start_position		u(posSize)	posSize as specified in subclause 6.5.3.2.3 or 6.5.3.5.3, depending on parameters_update_flag field
extended_AU_end_position		u(posSize)	posSize as specified in subclause 6.5.3.2.3 or 6.5.3.5.3, depending on parameters_update_flag field
}			
}			
else {			
if (U_signature_flag) {			As specified in subclause 6.5.3.2 or 6.5.3.5, depending on parameters_update_flag field
num_signatures		u(16)	
for (i=0;i<num_signatures;i++) {			

Table 26 *(continued)*

Syntax	Key	Type	Remarks
if (!U_signature_constant_length) {			As specified in subclause 6.5.3.2 or 6.5.3.5, depending on parameters_update_flag field
U_signature_length[i]		u(8)	
}			
U_cluster_signature[i]		u(U_signature_size)	U_signature_size inferred as specified in the U_cluster_signature semantics in subclause 6.5.4.3.3
}			
}			
}			
}			
}			
while(!byte_aligned( ))			As specified in subclause 6.2
nesting_zero_bit	f(1)		
}			

6.5.4.3.3 Semantics

access_unit_ID is the identifier of the access unit among all access units with same AU_type, and, if AU_type is not equal to U_TYPE_AU, with the same sequence_ID.

num_blocks is the number of blocks, as specified in subclause 6.5.5, in the access unit.

parameter_set_ID is a unique identifier, in the dataset containing this access unit, of the dataset parameter set at the lowest level in the hierarchy of dataset parameter sets, which shall be returned by a decapsulator compliant to this document along with the decapsulated access unit, as specified in subclause 6.5.4. Such hierarchy of dataset parameter sets is enabled by the parent_parameter_set_ID and parameter_set_ID fields of the dataset parameter set, as specified in subclause 6.5.3.5.3.

AU_type identifies the type of access unit and the type of data (class) carried therein as specified in Table 3 in subclause 5.3.

reads_count is a counter of the genomic sequence reads encoded in the access unit.

mm_threshold specifies the maximum number of substitutions a read (of class N or M) shall contain to be counted by mm_count. If set to 0 the feature of counting substitutions in encoded reads is disabled as no reads would be below threshold.

mm_count specifies the number of reads encoded in the access unit containing a number of substitutions which is equal to or lower than the threshold specified by mm_threshold. It shall always be set to 0 if the threshold is set to 0.

ref_sequence_id in case of access unit carrying (part of) a reference sequence (dataset_type, as specified in subclause 6.5.3.2), is the identifier of such reference sequence.

ref_start_position: in case of an access unit carrying (part of) a reference sequence (dataset_type, as specified in subclause 6.5.3.2), it specifies the position on the reference sequence of the left-most nucleotide encoded in this access unit.

ref_end_position: in case of an access unit carrying (part of) a reference sequence (dataset_type, as specified in subclause 6.5.3.2), it specifies the position on the reference sequence of the right-most nucleotide encoded in this access unit.

sequence_ID is the identifier of the reference sequence this access unit refers to. It shall be equal to one of the values of the seq_ID field listed in the dataset header, as described in subclause 6.5.3.2.

AU_start_position is the position of the left-most mapped base among the first alignments of all genomic records encoded in the access unit irrespective of the strand.

AU_end_position is the position of the right-most mapped base among the first alignments of all genomic records encoded in the access unit irrespective of the strand.

extended_AU_start_position specifies the position of the left-most mapped base among all alignments of all genomic records contained in the access unit, irrespective of the strand.

extended_AU_end_position specifies the position of the right-most mapped base among all alignments of all genomic records contained in the access unit, irrespective of the strand.

num_signatures is the number of signatures in the access unit.

U_signature_length is the length of one cluster signature as number of nucleotides.

U_cluster_signature is the signature of the cluster this access unit belongs to. The length in bits of this field, named U_signature_size in Table 26, shall be calculated as follows:

$$\text{U\_signature\_size} = \text{signature\_length} * \text{bits\_per\_symbol}$$

with bits_per_symbol as specified in Table 21, and signature_length corresponding either to U_signature_length as specified in subclause 6.5.3.2.2 when U_signature_constant_length (as specified in subclause 6.5.3.2.2) is equal to 1 or to the signature-specific U_signature_length[i] as specified in Table 26, when U_signature_constant_length (specified in subclause 6.5.3.2.2) is equal to 0.

The j-th base in a signature shall be inferred as follows:

$S_{\text{alphabet\_ID}}$[(U_cluster_signature[i] >> ((signature_length – j – 1) * bits_per_symbol)) & ((1 << bits_per_symbol) - 1)]

with alphabet_id as specified in subclause 6.5.3.2.2 and $S_{\text{alphabet\_ID}}$ as specified in subclause 6.5.3.2.4.

6.5.4.4 Access unit information

6.5.4.4.1 General

This is an optional box containing accessory information associated to the access unit.

6.5.4.4.2 Syntax

Table 27 — Access unit information syntax

Syntax	Key	Type	Remarks
AU_information {	*auin*		
if (minor_version != '1900') {			minor_version as specified in subclause 6.5.1
dataset_group_ID		u(8)	
dataset_ID		u(16)	
}			
AU_information_value()			As specified in ISO/IEC 23092-3.
}			

6.5.4.4.3 Semantics

AU_information_value() contains accessory information related to the access unit, as specified in ISO/IEC 23092-3.

6.5.4.5 Access unit protection

6.5.4.5.1 General

This is an optional box containing protection information associated to the access unit.

When present this box contains information that a decoder needs to properly handle a protected access unit.

6.5.4.5.2 Syntax

Table 28 — Access unit protection syntax

Syntax	Key	Type	Remarks
AU_protection {	*aupr*		
if (minor_version != '1900') {			minor_version as specified in subclause 6.5.1
dataset_group_ID		u(8)	
dataset_ID		u(16)	
}			
AU_protection_value()			As specified in ISO/IEC 23092-3
}			

6.5.4.5.3 Semantics

AU_protection_value() contains the access unit protection information, as specified in ISO/IEC 23092-3.

6.5.5 Block

6.5.5.1 General

A block is composed of a block header (as specified in subclause 6.5.5.2) and a block payload, containing compressed descriptors of the same type (descriptor_ID) and class (class_ID). In DSC mode, as specified in subclause 6.5.3.2.5, only the block payload is present in the descriptor stream, as specified in subclause 6.6.3.

Table 29 — Block syntax

Syntax	Key	Type	Remarks
block {			
block_header			As specified in subclause 6.5.5.2
for (i=0;i<block_payload_size;i++) {			block_payload_size as specified in subclause 6.5.5.2
block_payload[i]		u(8)	
}			
}			

block_payload[i] is the i-th byte of block payload, which contains compressed descriptors of the same type (descriptor_ID) and class (class_ID).

6.5.5.2 Block header

6.5.5.2.1 General

This box contains information associated to the block.

This box shall replace the block header provided by the underlying codec and specified in ISO/IEC 23092-2.

6.5.5.2.2 Syntax

Table 30 — Block header syntax

Syntax	Key	Type	Remarks
block_header {			
reserved		u(1)	
descriptor_ID		u(7)	
reserved		u(3)	
block_payload_size		u(29)	
}			

6.5.5.2.3 Semantics

descriptor_ID is the descriptor identifier, as specified in ISO/IEC 23092-2.

block_payload_size is the number of bytes composing the block payload.

6.6 Data structures specific to file format

6.6.1 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.

6.6.2 Indexing

6.6.2.1 Master index table

6.6.2.1.1 General

The master index table provides the indexing information needed to perform selective access on specific parts of the dataset.

It is present in the dataset when MIT_flag, as specified in subclause 6.5.3.2, is equal to 1. It is not present otherwise.

The first part of the master index table shall be ordered by increasing AU_start_position[seq_ID][ci][au_id] values; the second part of the master index table shall be ordered by increasing U_cluster_signature[uau_id][0] values.

The special value ((1<<byteOffsetSize)-1) assigned to AU_byte_offset[seq_ID][ci][au_id] represents an empty access unit. It is used to maintain synchronization among access units having different AU_type but covering the same genomic range.

The special value ((1<<byteOffsetSize)-1) assigned to block_byte_offset[seq_ID][ci][au_id][di] represents an empty block. It is used to maintain synchronization among blocks belonging to the same access unit.

6.6.2.1.2 Syntax

Table 31 — Master index table syntax

Syntax	Key	Type	Remarks
master_index_table {	*mitb*		
`for (seq=0;seq<seq_count;seq++) {`			seq_count as specified in subclause 6.5.3.2
`for (ci=0;ci<num_classes;ci++) {`			num_classes as specified in subclause 6.5.3.2
`if (clid[ci] != CLASS_U) {`			CLASS_U value as specified in subclause 5.2. clid as specified in subclause 6.5.3.2.
`for (au_id=0;au_id<seq_blocks[seq];au_id++) {`			seq_blocks as specified in subclause 6.5.3.2
`AU_byte_offset[seq][ci][au_id]`		u(byteOffsetSize)	byteOffsetSize as specified in subclause 6.5.3.2
`AU_start_position[seq][ci][au_id]`		u(posSize)	posSize as specified in subclause 6.5.3.2.3

Table 31 *(continued)*

Syntax	Key	Type	Remarks
AU_end_position[seq][ci][au_id]		u(posSize)	posSize as specified in subclause 6.5.3.2.3
if (dataset_type == 2) {			As specified in subclause 6.5.3.2
ref_sequence_id[seq][ci][au_id]		u(16)	
ref_start_position[seq][ci][au_id]		u(posSize)	posSize as specified in subclause 6.5.3.2.3
ref_end_position[seq][ci][au_id]		u(posSize)	posSize as specified in subclause 6.5.3.2.3
}			
if (multiple_alignment_flag) {			As specified in subclause 6.5.3.2
extended_AU_start_position[seq][ci][au_id]		u(posSize)	posSize as specified in subclause 6.5.3.2.3
extended_AU_end_position[seq][ci][au_id]		u(posSize)	posSize as specified in subclause 6.5.3.2.3
}			
if (!block_header_flag) {			As specified in subclause 6.5.3.2
for(di=0;di<num_descriptors[ci];di++) {			num_descriptors as specified in subclause 6.5.3.2
block_byte_offset[seq][ci][au_id][di]		u(byteOffsetSize)	byteOffsetSize as specified in subclause 6.5.3.2
}			
}			
}			
}			
}			
}			
for (uau_id=0;uau_id<num_U_access_units;uau_id++) {			
AU_byte_offset[uau_id]		u(byteOffsetSize)	
if(dataset_type == 2) {			As specified in subclause 6.5.3.2
U_ref_sequence_id[uau_id]		u(16)	
U_ref_start_position[uau_id]		u(posSize)	posSize as specified in subclause 6.5.3.2.3
U_ref_end_position[uau_id]		u(posSize)	posSize as specified in subclause 6.5.3.2.3
}			

Table 31 *(continued)*

Syntax	Key	Type	Remarks
`else {`			
`if(U_signature_flag != 0) {`			As specified in subclause 6.5.3.2
`num_signatures`		u(16)	
`for (i=0;i<num_signatures;i++) {`			
`if (!U_signature_constant_length) {`			As specified in subclause 6.5.3.2
`U_signature_length[uau_id][i]`		u(8)	
`}`			
`U_cluster_signature[uau_id][i]`		u(U_signature_size)	U_signature_size inferred as specified in the U_cluster_signature semantics in subclause 6.6.2.1.3
`}`			
`}`			
`}`			
`}`			
`while( !byte_aligned( ) )`			As specified in subclause 6.2
`nesting_zero_bit`		f(1)	Equal to 0
`if(!block_header_flag) {`			As specified in subclause 6.5.3.2
`for (di=0;di<num_descriptors[num_classes-1];` `di++) {`			num_descriptors as specified in subclause 6.5.3.2
`block_byte_offset[uau_id][di]`		u(byteOffsetSize)	byteOffsetSize as specified in subclause 6.5.3.2
`}`			
`}`			
`}`			
`}`			

6.6.2.1.3 Semantics

AU_byte_offset is the byte offset of the first byte in the access unit, with respect to the first byte of the Value field of the dataset (dtcn) gen_info structure (0-based). It is equal to ((1<<byteOffsetSize)-1) if the access unit is empty: in such a case the fields AU_start_position, AU_end_position, extended_AU_start_position, extended_AU_end_position, ref_sequence_id, ref_start_position and ref_end_position shall be ignored.

AU_start_position is the position of the left-most mapped base among the first alignments of all genomic records encoded in the access unit irrespective of the strand. AU_start_position[seq][ci][i+1] shall always be greater than or equal to AU_start_position[seq][ci][i].

AU_end_position is the position of the right-most mapped base among the first alignments of all genomic records encoded in the access unit irrespective of the strand.

ref_sequence_id: in case of an access unit carrying (part of) a reference sequence, it specifies the ID of such reference sequence.

ref_start_position: in case of an access unit carrying (part of) a reference sequence, it specifies the position on the reference sequence of the first nucleotide encoded in this access unit.

ref_end_position: in case of an access unit carrying (part of) a reference sequence, it specifies the position on the reference sequence of the last nucleotide encoded in this access unit.

extended_AU_start_position specifies the position of the left-most mapped base among all alignments of all genomic records contained in the access unit, irrespective of the strand.

extended_AU_end_position specifies the position of the right-most mapped base among all alignments of all genomic records contained in the access unit, irrespective of the strand.

block_byte_offset is the byte offset of the first byte in the block, with respect to the first byte of the Value field of the dataset (dtcn) gen_info structure (0-based). If the block is empty and block_header_flag is equal to 1, it shall be equal to ((1 << byteOffsetSize) – 1). If the block is empty and block_header_flag is equal to 0, it shall be equal either to the block_byte_offset value of the next block in the descriptor stream or, for the last block in the descriptor stream, to the sum of the block_byte_offset value of the first block in the descriptor stream and the descriptor stream payload size.

NOTE The descriptor stream payload size can be inferred as the L field of the dscn gen_info header, minus the L field of the dshd gen_info header, minus the L field of the dspr gen_info header.

num_U_access_units is the total number of access units in the dataset containing encoded data of class U. It is encoded in the dataset header, as specified in subclause 6.5.3.2.

U_ref_sequence_id, in case of an access unit carrying (part of) a reference sequence, is the identifier of such reference sequence.

U_ref_start_position, in case of an access unit carrying (part of) a reference sequence, specifies the position on the reference sequence of the left-most nucleotide encoded in this access unit.

U_ref_end_position, in case of an access unit carrying (part of) a reference sequence, specifies the position on the reference sequence of the right-most nucleotide encoded in this access unit.

num_signatures is the number of signatures.

U_signature_length[uau_id][i] is the length of cluster signatures as number of nucleotides.

U_cluster_signature[uau_id][i] is the i-th signature of the cluster the access unit belongs to. U_cluster_signature[uau_id][i+1] shall always be greater than or equal to U_cluster_signature[uau_id][i]. The length in bits of this field, named U_signature_size in Table 26, shall be calculated as follows:

U_signature_size = signature_length * bits_per_symbol

with bits_per_symbol as specified in Table 21, and with signature_length corresponding either to U_signature_length as specified in subclause 6.5.3.2.2 when U_signature_constant_length (as specified in subclause 6.5.3.2.2) is equal to 1 or to the signature-specific U_signature_length[uau_id][i] as specified in Table 31 when U_signature_constant_length (as specified in subclause 6.5.3.2.2) is equal to 0.

The j-th nucleotide in a signature shall be inferred as follows:

$S_{\text{alphabet\_ID}}$[(U_cluster_signature[i] >> ((signature_length – j – 1) * bits_per_symbol)) & ((1 << bits_per_symbol) - 1)]

with alphabet_id as specified in subclause 6.5.3.2.2 and $S_{\text{alphabet\_ID}}$ as specified in subclause 6.5.3.2.4.

6.6.3 Descriptor stream

6.6.3.1 General

A descriptor stream is a stream of data of a certain class and descriptor, encoded as described in ISO/IEC 23092-2.

This is a mandatory box when the syntax element block_header_flag in the dataset header, as specified in subclause 6.5.3.2, is equal to 0; it is forbidden otherwise.

Child boxes may be present or not, according to the column “Mandatory” in Table 4.

Table 32 — Descriptor stream syntax

Syntax	Key	Type	Remarks
descriptor_stream {	*dscn*		
descriptor_stream_header	dshd	gen_info	As specified in subclause 6.6.3.2
DS_protection	dspr	gen_info	As specified in subclause 6.6.3.3
for (i=0;i<num_blocks;i++) {			num_blocks as specified in subclause 6.6.3.2.3
for (j=0;j<block_payload_size[i];j++) {			
block_payload[i][j]		u(8)	As specified in ISO/IEC 23092-2
}			
}			
}			

block_payload_size[i] is inferred from the master index table field block_byte_offset, as specified in subclause 6.6.2.1, as difference between either block_byte_offset[i+1] or the variable descriptor_stream_size, as specified in subclause 6.5.3.2.5, and block_byte_offset[i].

block_payload[i][j] is the j-th byte of the block payload.

6.6.3.2 Descriptor stream header

6.6.3.2.1 General

This is a box describing a descriptor stream. It is mandatory whenever the descriptor stream, as specified in subclause 6.6.3, is present, forbidden otherwise.

6.6.3.2.2 Syntax

Table 33 — Descriptor stream header syntax

Syntax	Key	Type	Remarks
descriptor_stream_header {	*dshd*		
reserved		u(1)	
descriptor_ID		u(7)	
class_ID		u(4)	
num_blocks		u(32)	
while(!byte_aligned( ))			As specified in subclause 6.2
nesting_zero_bit		f(1)	Equal to 0
}			

6.6.3.2.3 Semantics

descriptor_ID identifies the type of compressed descriptors carried by the descriptor stream. It shall have one of the values specified as descriptor_ID[ci][di] in subclause 6.5.3.2.

class_ID identifies the class of data carried by the block, as specified in Table 2.

num_blocks is the number of blocks composing the descriptor stream.

6.6.3.3 Descriptor stream protection

6.6.3.3.1 General

This is an optional box containing protection information associated to a descriptor stream.

When present this box contains information that a decoder needs to properly handle a protected descriptor stream.

6.6.3.3.2 Syntax

Table 34 — Descriptor stream protection syntax

Syntax	Key	Type	Remarks
DS_protection {	*dspr*		
DS_protection_value()			As specified in ISO/IEC 23092-3
}			

6.6.3.3.3 Semantics

DS_protection_value(): descriptor stream protection information, specified in ISO/IEC 23092-3.

6.6.4 Offset

This box allows an indirect addressing of boxes in a different physical position in the file, while preserving their logical position as described in this document. It shall be placed in the mandatory position of the addressed box, as specified in subclause 6.1.2, so that the logical position of the addressed box would still be respecting such mandatory ordering.

In case of boxes not marked with suffix “[]” after their name in the Syntax column of any of the tables in subclauses 6.5 and 6.6, and which can be present in only one instance, if an associated offset box is present then multiple instances of the same original box may be physically present in the File, but only the box addressed by the offset box shall be considered as valid, while the other instances of the same box shall be ignored.

In case of boxes marked with suffix “[]” after their name in the Syntax column of any of the tables in subclauses 6.5 and 6.6, and which may be present in multiple instances, if the associated offset box is present it shall be present in as many instances as the addressed boxes.

In case one instance of the offset box is not referring to any box yet but just present as placeholder for a new box which may potentially be added, then the Offset field, as specified in subclause 6.6.4.1, shall take the (1<<64)–1 value, which shall be ignored.

6.6.4.1 Syntax

```
struct offset
{
        c(4)            Key;
        c(4)            SubKey;
        u(64)           Offset;
}
```

6.6.4.1.1 Semantics

Key is the key of the offset box, being equal to *offs*.

SubKey is the key of the box being addressed by the offset box. Its usage is restricted to the following boxes, as specified in Table 4: dghd, rfgn, rfmd, labl, lbll, dgmd, dgpr, dtcn, dtmd, dtpr.

Offset is the byte offset of the first byte in the referenced box, with respect to the first byte of the file (0-based). If equal to (1<<64)–1 then the offset box is not addressing any box and shall be ignored. The value of Offset shall be larger than the offset of the last byte of any dgcn box in the file.

6.7 Data structures specific to transport format

6.7.1 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.

6.7.2 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, and it can transport any of the following data structures:

- file_header, as specified in subclause 6.5.1: 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) equal to 1,
- data structures containing transport information (dataset mapping table list as specified in subclause 6.7.3, dataset mapping table as specified in subclause 6.7.4),
- dataset group header, as specified in subclause 6.5.2.2,
- reference, as specified in subclause 6.5.2.3,
- label list, as specified in subclause 6.5.2.5,
- dataset header, as specified in subclause 6.5.3.2,
- dataset parameter set, as specified in subclause 6.5.3.3,
- access unit, as specified in subclause 6.5.4,
- 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.

6.7.3 Dataset mapping table list

6.7.3.1 General

This is a mandatory box containing a list of all Stream_IDs of data streams transporting the dataset mapping tables, as specified in subclause 6.7.4, available in the datasets of a dataset group. Each of the listed data streams is identified by a unique dataset_mapping_table_SID.

The dataset mapping table list contains, along with the dataset mapping table described in subclause 6.7.4, the necessary and sufficient information to de-packetize and de-capsulate the transport format for a specific dataset in a dataset group.

Each dataset mapping table list is transported within one or more packets with Stream ID (SID in packet header, as specified in subclause 6.7.5.2) equal to 0.

6.7.3.2 Syntax

Table 35 — Dataset mapping table list syntax

Syntax	Key	Type	Remarks
dataset_mapping_table_list {	*dmtl*		
`dataset_group_ID`		u(8)	
`for (i=0;i<num_datasets;i++) {`			
`dataset_mapping_table_SID`		u(16)	
`}`			
}			

6.7.3.3 Semantics

dataset_group_ID is the dataset group ID, as in the dataset group header, as specified in subclause 6.5.2.2.

num_datasets is inferred from the Length field in dataset_mapping_table_list *gen_info* header,as follows: (Length – 13) / 2.

dataset_mapping_table_SID is the Stream ID associated to the data stream containing the dataset_mapping_table. Values 0 and 1 cannot be used as reserved for the dataset mapping table list, as specified in subclause 6.7.3, and the file header, as specified in subclause 6.5.1. The same value of dataset_mapping_table_SID cannot be contained in dataset_mapping_tables with different dataset_group_ID.

6.7.4 Dataset mapping table

6.7.4.1 General

This is a mandatory box listing all data streams transporting data related to the dataset identified by dataset_ID.

The dataset mapping table associates data types (access units, metadata boxes, protection boxes, etc.) and Stream IDs (SID) found in packet header, as specified in subclause 6.7.5.2.

Table 37 provides the association between data type values and data structures. The dataset mapping table contains, along with the dataset mapping table list specified in Table 35, the necessary and sufficient information to de-packetize and de-capsulate the transport format for a dataset in a dataset group.

The datasets mapping table can be periodically re-transmitted, either updated or identical.