



**International
Standard**

ISO/IEC 23092-1

**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

**Third edition
2025-01**

Reference number
ISO/IEC 23092-1:2025(en)

Full document details: [ISO_IEC 23092-1_2025](#)


© ISO/IEC 2025

Documents
Document Sharing Hub



COPYRIGHT PROTECTED DOCUMENT

© ISO/IEC 2025

All rights reserved. Unless otherwise specified, or required in the context of its implementation, no part of this publication may be reproduced or utilized otherwise in any form or by any means, electronic or mechanical, including photocopying, or posting on the internet or an intranet, without prior written permission. Permission can be requested from either ISO at the address below or ISO's member body in the country of the requester.

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

Page

Foreword	v
Introduction	vii
1 Scope	1
2 Normative references	1
3 Terms and definitions	1
4 Conventions	4
4.1 Operators and functions	4
4.1.1 Arithmetic operators	4
4.1.2 Logical operators	4
4.1.3 Relational operators	4
4.1.4 Bitwise operators	4
4.1.5 Assignment operators	5
4.1.6 String/Character functions and operator	5
4.1.7 Data structure function and operator	5
4.1.8 Mathematical functions	5
4.1.9 Array operation functions	5
4.2 Syntax and semantics	6
4.2.1 Method of specifying syntax in tabular form	6
4.2.2 Bit ordering	6
4.2.3 Specification of syntax functions	6
4.2.4 Processes	7
5 Structure of coded genomic data	7
5.1 Genomic sequencing data record	7
5.2 Genomic annotation data records	8
5.3 Data classes	9
5.4 Access units	10
5.5 Datasets	10
5.6 Annotation data tile	11
5.7 Annotation tables	11
5.8 Annotation access units	11
5.9 Selective access	12
6 Data format	12
6.1 Format structure	12
6.1.1 General	12
6.1.2 Box order	17
6.2 Syntax for representation	18
6.3 Output data unit	19
6.4 Data structures common to file format and transport format	20
6.4.1 File header	20
6.4.2 Dataset group	20
6.4.3 Dataset	29
6.4.4 Access unit	40
6.4.5 Block	46
6.4.6 Annotation Table	47
6.4.7 Attribute Group	57
6.4.8 Annotation access unit	59
6.4.9 AAU block	63
6.5 Data structures specific to file format	64
6.5.1 General	64
6.5.2 Indexing	64
6.5.3 Descriptor stream	74
6.5.4 Offset	76
6.6 Data structures specific to transport format	77

6.6.1	General	77
6.6.2	Data streams	77
6.6.3	Dataset mapping table list	77
6.6.4	Dataset mapping table	78
6.6.5	Packet	80
6.7	Reference procedures to convert transport format to file format	81
6.7.1	Procedure for genomic sequencing data	81
6.7.2	Procedure for genomic annotation data	83
7	String indexing technologies	87
7.1	Master string index	87
7.1.1	General	87
7.1.2	Syntax	87
7.1.3	Master String Index Header	87
7.1.4	String index	88
7.1.5	Compressed string index	90
7.2	Decoding and querying processes	96
7.2.1	String index payload	96
7.2.2	Helper functions	97
7.2.3	Substring decoding process	98
7.2.4	Suffix array lookup process	99
7.2.5	Inverse suffix array process	99
7.2.6	Character decoding process	100
7.2.7	LF-mapping process	101
7.2.8	Extended LF-mapping process	101
7.2.9	Substring position search process	102
7.2.10	Searching for substring positions with the string index	103
7.2.11	Decoding a subset of the string index	104
7.2.12	Decoding all the strings of a specific annotation data tile	104
7.2.13	Retrieving whole strings with the string index	106
7.2.14	Retrieving data tile index(es) associated with a position and record indexes	107
8	Indexing for numeric range searches	110
8.1	B-Tree indexing	110
8.1.1	General	110
8.1.2	Syntax	110
8.1.3	Semantics	111
Annex A (informative) IETF RFC 3986 specification summary		112
Annex B (informative) Selective access strategies for genomic sequencing data		113
Annex C (informative) Selective access strategies for genomic annotation data		116
Annex D (informative) Depacketization process		132
Annex E (informative) Efficient handling of symmetric annotation data		135
Bibliography		137

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 or www.iec.ch/members_experts/refdocs).

ISO and IEC draw attention to the possibility that the implementation of this document may involve the use of (a) patent(s). ISO and IEC take no position concerning the evidence, validity or applicability of any claimed patent rights in respect thereof. As of the date of publication of this document, ISO and IEC had received notice of (a) patent(s) which may be required to implement this document. However, implementers are cautioned that this may not represent the latest information, which may be obtained from the patent database available at www.iso.org/patents and <https://patents.iec.ch>. ISO and IEC shall not be held responsible for identifying any or all such patent rights.

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. In the IEC, see www.iec.ch/understanding-standards.

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 third edition cancels and replaces the second edition (ISO/IEC 23092-1:2020), which has been technically revised.

The main changes are as follows:

- Updates to the overall hierarchy of data structures and box order in [subclause 6.1](#)
- Extensions for the transport and storage of genomic annotation data, in addition to genomic sequencing data, in support of ISO/IEC 23092-6:2023 specifications while maintaining backward compatibility, which include:
 - An overview of genomic annotation data records in [subclause 5.2](#), with detailed formats specified in Part 6
 - Basic annotation table information in dataset header (as specified in [subclause 6.4.3.2](#)) and annotation encoding parameters in dataset parameter set (as specified in [subclause 6.4.3.7](#))
 - Additional data structures such as annotation table (atcn, as specified in [subclause 6.4.6](#)), attribute group (agcn, as specified in [subclause 6.4.7](#)), annotation access unit (aauc, as specified in [subclause 6.4.8](#)), AAU block (as specified in [subclause 6.4.9](#)), attribute data byte offset (adbo, as specified in [subclause 6.5.2.3](#)) and annotation table index (atix, as specified in [subclause 6.5.2.4](#))
 - The reference procedure for conversion from transport format to file format for genomic annotation data in [subclause 6.7.2](#)

- Data structure for B-Tree indexing (as specified in [subclause 8.1](#)) and selective access strategies for genomic annotation data (as specified in [Annex C](#))
- Extensions in support of ISO/IEC 23092-3:2022 which include:
 - New container boxes for metrics metadata: DT_metrics (dtmt, as specified in [subclause 6.4.3.4](#)) and AU_metrics (aamt, as specified in [subclause 6.4.4.5](#)), containing statistical information (with detailed formats specified in Part 3), which allows for fast and direct extraction of statistics associated with the dataset and access unit content
 - New container boxes for clinical data linkage (CDL) metadata: DG_CDL (dgcd, as specified in [subclause 6.4.2.7](#)), DT_CDL (dtcd, as specified in [subclause 6.4.3.5](#)) and AT_CDL (atcd, as specified in [subclause 6.4.6.4](#)), for establishing linkages to external data sources, which enables access to the clinical data of individual samples
- The inclusion of FM-Index-based entropy coding algorithm (as specified in [Clause 7](#)), which provides string search capabilities in the compressed domain

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

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 and www.iec.ch/national-committees.

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.

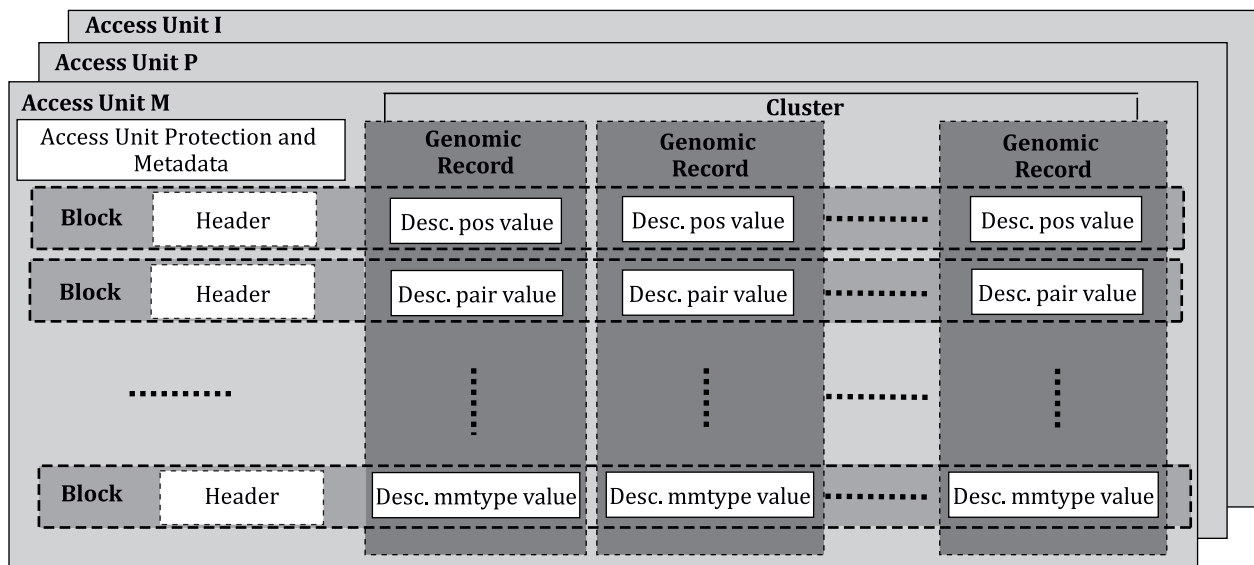


Figure 1 — Access units, blocks and genomic records

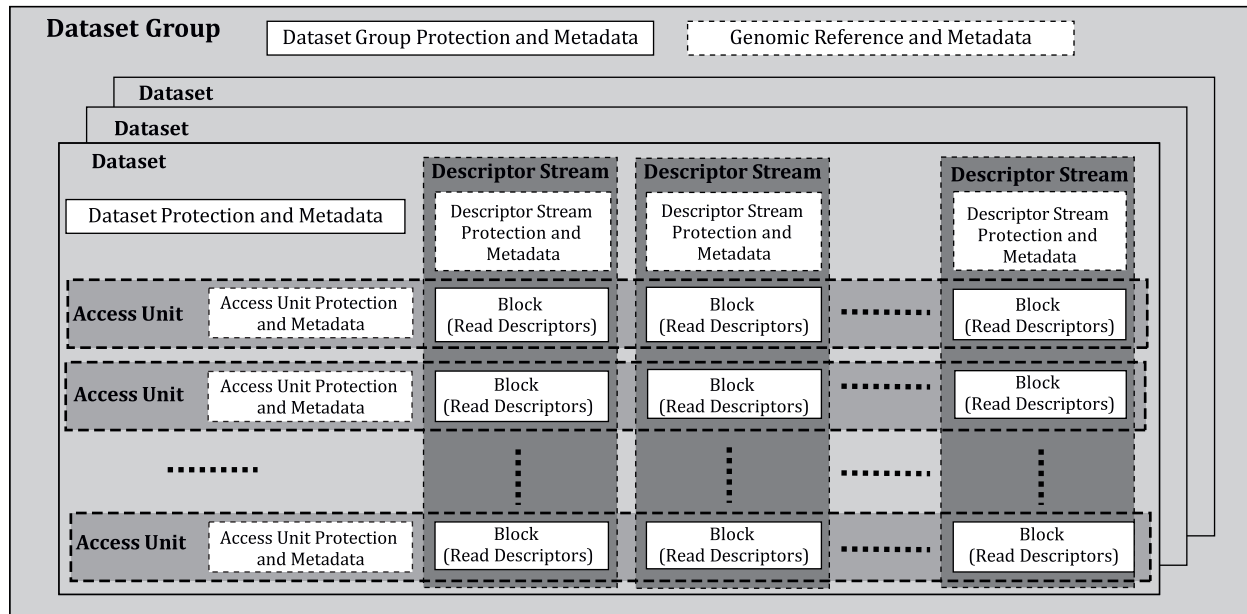


Figure 2 — High-level data structure: datasets and dataset group

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](#).

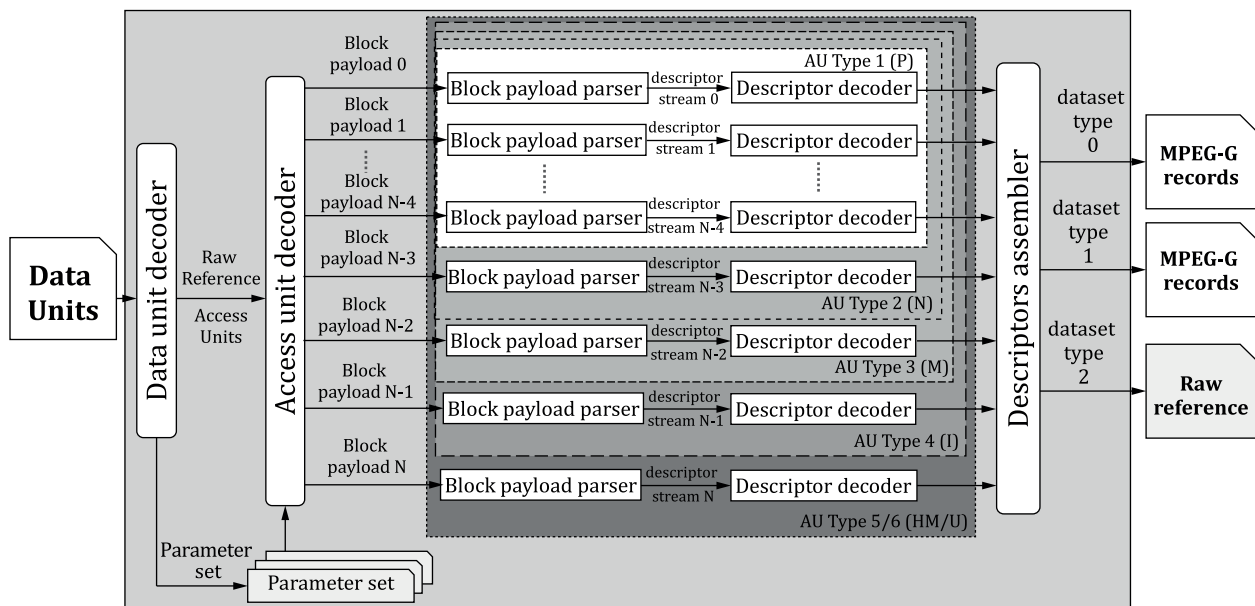


Figure 3 — Decoding process

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.

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)*

ISO/IEC 23092-6, *Information technology — Genomic information representation — Part 6: Coding of genomic annotations*

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 terminology databases for use in standardization at the following addresses:

- ISO Online browsing platform: available at <https://www.iso.org/obp>
- IEC Electropedia: available at <https://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

alignment

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

3.7

box

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

3.8

cluster

aggregation of genomic records

3.9

cluster signature

signature

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

3.10

container box

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

3.11

data stream

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

3.12

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

3.13

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

3.14

file format

set of data structures for the storage of coded information

3.15

genomic position

position

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

3.16

genomic region

region

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

3.17

genomic range

range

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

3.18

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

3.19

packet

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

3.20

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.

3.21

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.

3.22

genomic segment

segment

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

3.23

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

3.24

syntax field

element of data represented in the data format

3.25

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.

3.26

transport format

set of data structures for the transport of coded information

3.27**variable**

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

4 Conventions**4.1 Operators and functions**

NOTE The 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.

4.1.1 Arithmetic operators

+	addition
–	subtraction (as a binary operator) or negation (as a unary operator)
*	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)

4.1.2 Logical operators

	logical OR
&&	logical AND
!	logical NOT
$x ? y : z$	If x is TRUE or not equal to 0, evaluates to the value of y ; otherwise, evaluates to the value of z .

4.1.3 Relational operators

>	greater than
≥	greater than or equal to
<	less than
≤	less than or equal to
==	equal to
!=	not equal to

4.1.4 Bitwise operators

&	AND
	OR
>>	shift right with sign extension
<<	shift left with 0 fill

4.1.5 Assignment operators

The following arithmetic operators are defined as follows:

=	assignment operator
++	increment, i.e., $x++$ is equivalent to $x = x + 1$; when used in an array index, evaluates to the value of the variable prior to the increment operation.
--	decrement, i.e., $x--$ is equivalent to $x = x - 1$; when used in an array index, evaluates to the value of the variable prior to the decrement operation.
+=	increment by amount specified, i.e., $x += 3$ is equivalent to $x = x + 3$, and $x += (-3)$ is equivalent to $x = x + (-3)$.
-=	decrement by amount specified, i.e., $x -= 3$ is equivalent to $x = x - 3$, and $x -= (-3)$ is equivalent to $x = x - (-3)$.

4.1.6 String/Character functions and operator

+	string concatenation, when applied on variables of string or character types
---	--

4.1.7 Data structure function and operator

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

`N->m` returns the data field m defined within the data structure N

4.1.8 Mathematical functions

The following mathematical functions are defined:

`Ceil( x )` the smallest integer greater than or equal to x

`Floor( x )` the largest integer less than or equal to x

`Log( x )` the base-e logarithm of x

`Log2( x )` the base-2 logarithm of x

`Min( x, y )` $\begin{cases} x & ; \ x \leq y \\ y & ; \ x > y \end{cases}$

`Max( x, y )` $\begin{cases} x & ; \ x \geq y \\ y & ; \ x < y \end{cases}$

4.1.9 Array operation functions

`array_dims[] = Size(ndimensional_array[...]..., dim)` returns the dimensional size(s) of variable `ndimensional_array`. If `dim` is not specified, it returns a vector with the i^{th} element corresponding to the size of the i^{th} dimension. If `dim` is specified, it returns only the size of the dimension corresponding to `dim`, with 1 being the first dimension.

`Max(x[])` returns the maximum value in a numeric array `x[]`.

`Find(x[], v)` returns the index of the first element in array `x[]` that is equal to the value `v`. It returns -1 if `v` is not found.

4.2 Syntax and semantics

4.2.1 Method of specifying syntax in tabular form

[Table 1](#) 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 1 — Constructs used to express the conditions when data elements are present

Construct	Description
<pre>if (condition) { data_element . . . }</pre>	If the condition is true, then the first group of data elements occurs next in the bitstream.
<pre>else { data_element . . . }</pre>	If the condition is not true, then the second group of data elements occurs next in the bitstream.
<pre>for (i=0; i<n; i++) { data_element . . . }</pre>	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 2](#).

Table 2 — Syntax used to represent collections of data elements

Syntax	Description
<code>data_element[]</code>	<code>data_element[]</code> is an array of data. The number of data elements is indicated by the semantics.
<code>data_element[n]</code>	<code>data_element[n]</code> is the n^{th} element of an array of data.
<code>data_element[m][n]</code>	<code>data_element[m][n]</code> is the m^{th} , n^{th} element of a two-dimensional array of data.
<code>data_element[l][m][n]</code>	<code>data_element[l][m][n]</code> is the l^{th} , m^{th} , n^{th} element of a three-dimensional array of data.

4.2.2 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).

4.2.3 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 $(\text{stringLength} + 1) * 8$ 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.

4.2.4 Processes

Processes are used to describe the sequence of operations on syntax elements (see example in [Table 3](#)). A process has a separate specification and unique invocation. All syntax elements and variables that pertain to the current syntax structure and dependent syntax structures are available in the process specification and at the time of invocation. A process specification may also have a lower-case variable explicitly specified as input. Each process specification output is explicitly defined. The output of a process can be assigned to one or many variables as specified in the return clause of the process.

Table 3 — Example decoding process

Syntax	Remarks
<code>process_xyz(counter) {</code>	A decoding process with one parameter
num	A syntax element
<code>for(i = 0; i < num * counter; i++) {</code>	
value[i]	A syntax element
<code>}</code>	
<code>a_sub_process</code>	A sub-decoding process within process_xyz
<code>}</code>	

5 Structure of coded genomic data

5.1 Genomic sequencing data record

The genomic sequencing data 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 4](#) 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 4 — Genomic sequencing data 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 4](#). These records shall be generated by decoders compliant to ISO/IEC 23092-2 as output of the decoding process.

5.2 Genomic annotation data records

Different types of genomic annotation data records are associated with each annotation table type (AT_type, as specified in [subclause 6.4.6.2](#)). They are encoded and stored in attribute groups of different classes (AG_class, as specified in [subclause 6.4.7.2](#)) in an annotation table. [Table 5](#) summarizes the annotation table types and their associated data records.

Table 5 — Genomic annotation data records

Annotation table type	Attribute group class	Record type	Record content
Variants	1	Variant site record	Variant site information including chromosome, position, reference allele, alternative allele, variant ID, quality value and other relevant attributes
	0	Variant genotype record	Variant genotype values for individual samples and any associated metrics
	2	Sample record	Sample information such as sample ID, demographics and clinical data
Functional annotations	0	Functional annotation record	Any number of ontological labels and attributes associated with a genomic feature specified by its genomic range, strand, name and ID
	5	Track property record	Track property containing display attributes associated with a block of functional annotation records referenced by the index of the first record in the block
Gene expression	0	Expression record	Expression values associated with a genomic feature, such as a gene or an exon, for individual samples
	1	Feature record	Feature information such as gene name and chromosome position
	2	Sample record	Sample information such as sample ID, demographics and clinical data
Contact matrices	0	Contact matrix record	The frequency of contact, measured by counts or any number of normalized values, in 3D space between any two genomic regions in a pair of the same or different chromosomes.
Tracks	0	Track data record	Any number of attributes associated with a genomic range and strand
	5	Track property record	Track property containing display attributes associated with a block of functional annotation records referenced by the index of the first record in the block

ISO/IEC 23092-6 defines the formats of the above output records. These records shall be generated by decoders compliant to ISO/IEC 23092-6 as output of the decoding process.

5.3 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 6](#).

Table 6 — 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 6.4.5](#). 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.

5.4 Access units

Access units (AUs) are data structures containing a coded representation of genomic sequencing data. Related metadata may be available to facilitate bitstream access and manipulation. An access unit contains either genomic sequencing 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, as specified in [Table 7](#), according to the class of the coded data.

Table 7 — 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.

5.5 Datasets

A dataset is a data structure containing a header, optional metadata, access units for genomic sequencing data, and annotation tables for genomic annotation data. The set of access units or annotation tables composing the dataset constitutes the dataset payload.

One or more datasets are assembled into a dataset group.

5.6 Annotation data tile

An annotation data tile specifies a rectangular region of descriptor/attribute data corresponding to contiguous ranges of rows and columns (if data is two-dimensional) of an annotation table. Instructions on how to divide descriptor/attribute data into tiles are specified in tile configurations. Each data tile per descriptor/attribute is individually encoded and stored in an annotation access unit payload block. Selective access to data in specific regions of interest is achieved by identifying and decoding only the data tiles that overlap with the regions of interest.

5.7 Annotation tables

An annotation table is a container for genomic annotation data of a designated annotation table type and subtype (the associated external file format), as specified in [Table 8](#) and [Table 9](#). It contains a header, optional metadata, tile configurations (for specifying how data shall be divided into tiles to be individually encoded), annotation table index (for supporting random access), and attribute groups. Each attribute group has a designated class that indicates the role of the descriptor/attribute data it contains. For example, an attribute group of class 0 contains main table data (such as variant genotypes), whereas class 1 contains auxiliary data associated with the rows of the main data (such as variant site information), and class 2 contains auxiliary data associated with the columns of the main data (such as sample information).

Table 8 — Annotation table type

Annotation table type		Data type
Annotation table type name	Value	
VARIANTS	1	Genomic variants
FUNCTIONAL_ANNOTATIONS	2	Functional annotations
GENE_EXPRESSION	3	Gene expression values
CONTACT_MATRICES	4	Position-to-position contact intensity values
TRACKS	5	Genome browser tracks

Table 9 — Annotation table sub-type

Annotation table type name	Annotation table sub-type	
	Annotation table sub-type name	Value
Any	UNDEFINED	0
VARIANTS	VCF	1
FUNCTIONAL_ANNOTATIONS	GTF	2
	GFF	3
	GENBANK	8
TRACKS	BED	4
	BEDGRAPH	5
	WIG	6
	BIGWIG	7
CONTACT_MATRICES	HIC	10
GENE_EXPRESSION	GENE_EXPRESSION	9

5.8 Annotation access units

An annotation access unit contains individually encoded payload blocks of genomic annotation data. Each payload block corresponds to a tile of data of a descriptor/attribute from specific ranges of rows and columns (if data is two-dimensional) in an annotation table. It supports two contiguity modes:

- Attribute contiguity, where each annotation access unit corresponds to a descriptor/attribute and each payload block within corresponds to a tile location in an annotation table, and

- Tile contiguity, where each annotation access unit corresponds to a tile location in an annotation table and each payload block within corresponds to a descriptor/attribute.

The annotation access unit is the smallest data organization handled by a decoder compliant with ISO/IEC 23092-6.

5.9 Selective access

For genomic sequencing data:

- 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](#).

For genomic annotation data:

- In the case of selective access to specific descriptor and attribute data in an annotation table region defined by ranges of indexes and/or genomic positions, the decoder shall identify the data tiles that overlap the region based on the relevant attribute data tile structure(s) as specified in [subclause 6.4.6.7](#) and/or genomic range index(es) as specified in [subclause 6.5.2.4.4](#), and return
 - a) if `attribute_contiguity == 0`, annotation access units corresponding to the identified data tiles and containing only the payload blocks of the selected descriptor(s) and attribute(s)
 - b) if `attribute_contiguity == 1`, annotation access units corresponding to the selected descriptor(s) and attribute(s) and containing only the payload blocks of the identified data tiles
 - c) the annotation parameter set(s) required for decoding the annotation access units identified in a) or b)
- Examples of selective access strategies are described in [Annex C](#).

6 Data format

6.1 Format structure

6.1.1 General

[Table 10](#) 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 for genomic sequencing data, genomic annotation data or both are marked respectively with a plus sign (+), pound sign (#) or 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 4-7](#). 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 10](#), the corresponding box is represented either with no Key and no Length, but only Value in the *gen_info* format, as specified in [subclause 6.2](#), for all boxes but offset, or as specified in [subclause 6.5.4.1](#) for the offset box.

Table 10 — Format structure and encapsulation levels

Box key (with hierarchical level)						Subclause	Scope	Mandatory
flhd						6.4.1		*
dgcn						6.4.2	File	*
	dghd					6.4.2.2		*
	rfgn					6.4.2.3		
	rfmd					6.4.2.4		
	labl					6.4.2.5		
	lbll					6.4.2.5.4		
	dgmd					6.4.2.6		
	dgcd					6.4.2.7		
	dgpr					6.4.2.8		
	dmtl					6.6.3.1	Transport	*
	dtcn					6.4.3	File	*
		dthd				6.4.3.2		*
		dtmd				6.4.3.3		
		dtmt				6.4.3.4		
		dtcd				6.4.3.5		
		dtpd				6.4.3.6		
		pars				6.4.3.7		*
		mitb				6.5.2.1	File	
		msix				7.1.2		
			mshd			7.1.3		
			ssix			7.1.4		
				csix		7.1.5		
		dmtb				6.6.4	Transport	*
		dscn				6.5.3	File	
			dshd			6.5.3.2	File	
			dspr			6.5.3.3	File	
		aucn				6.4.4		+
			auhd			6.4.4.3		+
			auin			6.4.4.4		
			aumt			6.4.4.5		
			aupr			6.4.4.6		
			<i>block</i>			6.4.5		+
				<i>block_header</i>		6.4.5.2		
		atcn				6.4.6	File	#
			athd			6.4.6.2		#
			atmd			6.4.6.3		
			atcd			6.4.6.4		
			atpr			6.4.6.5		
			agtc			6.4.6.6		#
				adts		6.4.6.7		#
				adbo		6.5.2.3	File	#
			atix			6.5.2.4	File	
				grix		6.5.2.4.4	File	

Table 10 (continued)

Box key (with hierarchical level)							Subclause	Scope	Mandatory
				avix			6.5.2.4.5	File	
			agcn				6.4.7	File	#
				aghd			6.4.7.2		#
				aauc			6.4.8		#
					aauh		6.4.8.3		#
					aaup		6.4.8.4		
					<i>aau_block</i>		6.4.9		#
						<i>aau_block_header</i>	6.4.9.2		#
	<i>offset</i>	<i>offset</i>					6.5.4	File	
<i>packet</i>							6.6.5	Transport	*
	<i>packet_header</i>						6.6.5.2	Transport	*

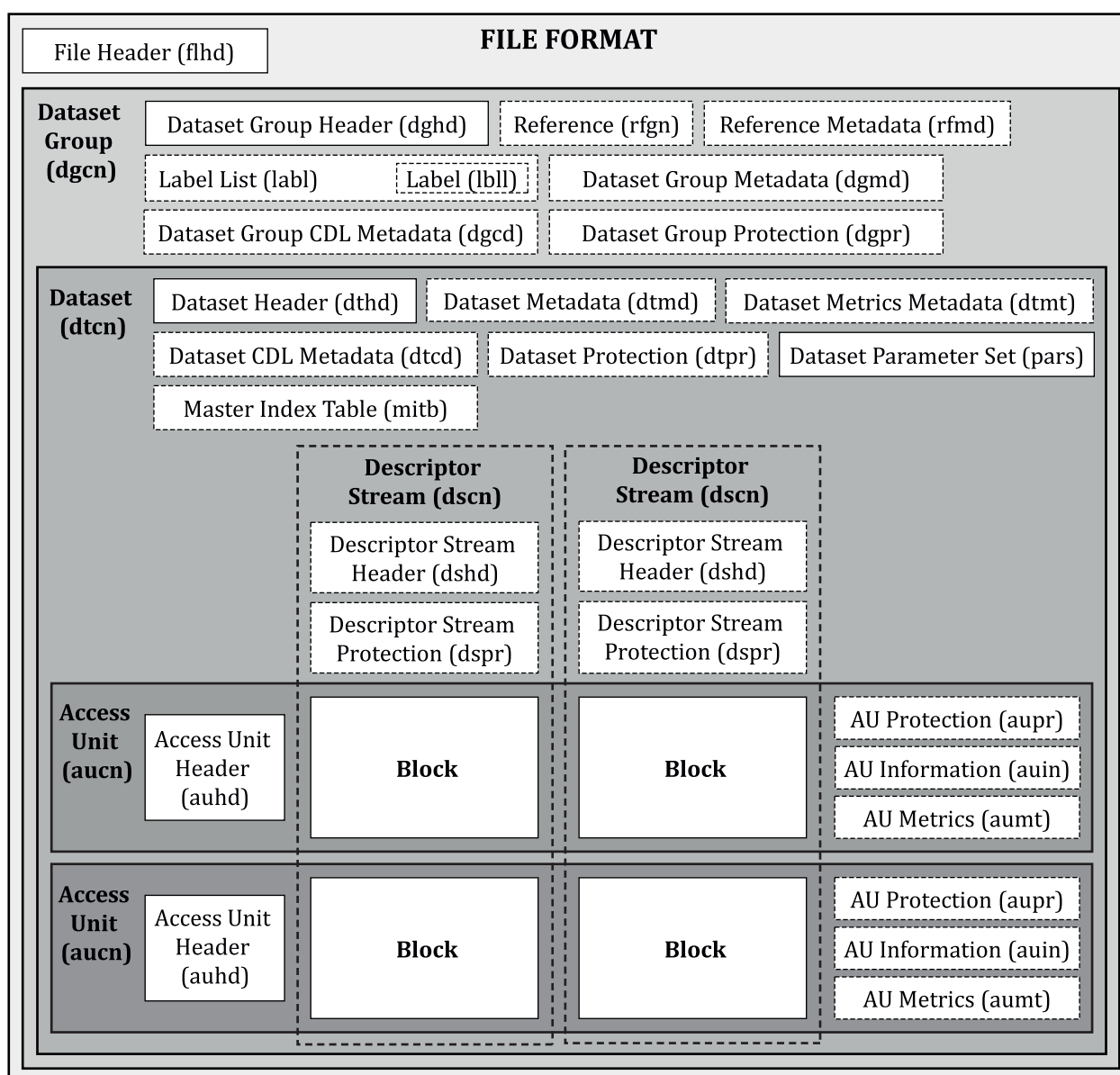


Figure 4 — Data structures hierarchy for storage of genomic sequencing data

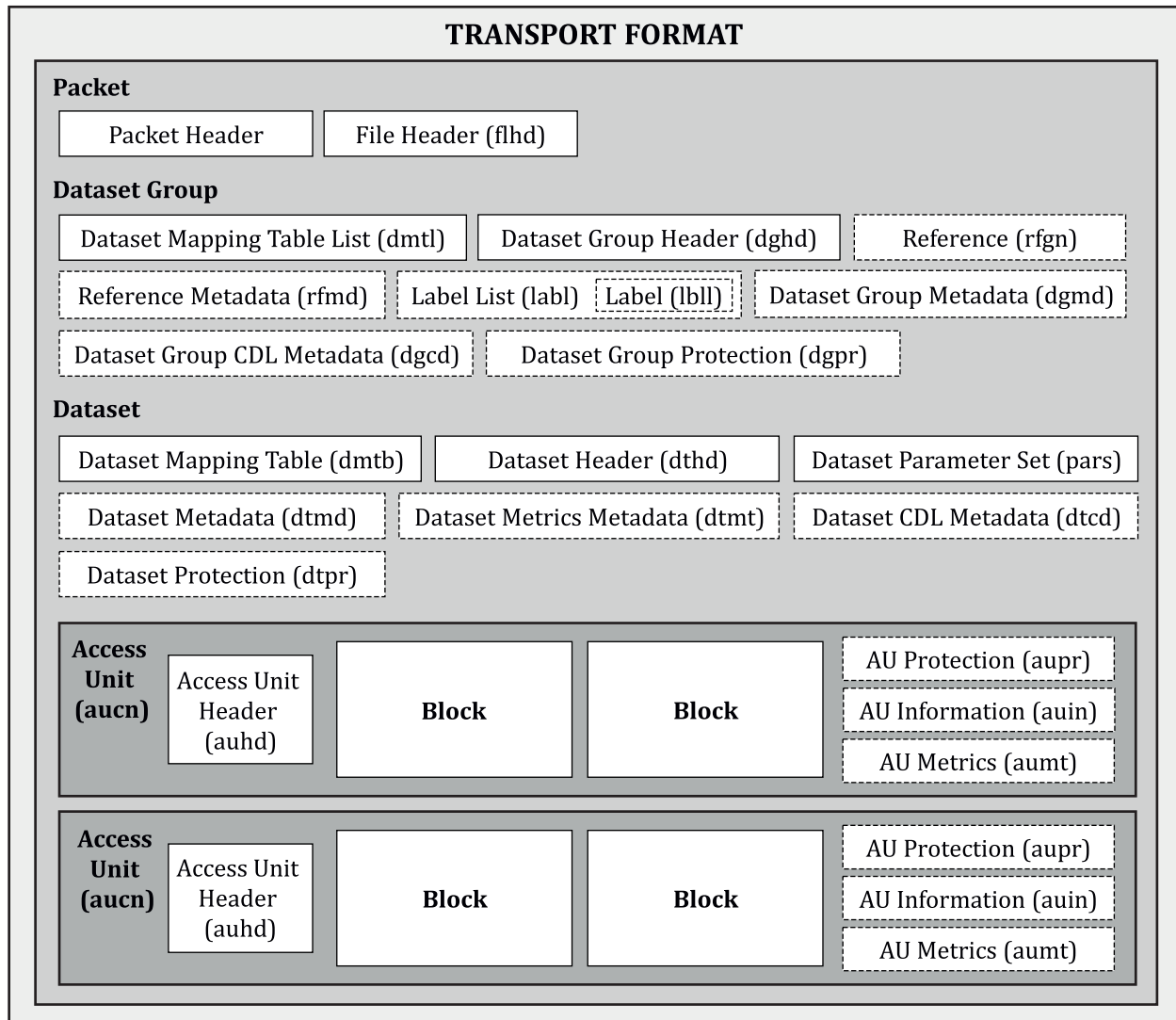


Figure 5 — Data structures hierarchy for transport of genomic sequencing data

FILE FORMAT

File Header (flhd)

Dataset Group (dgcn)

Dataset Group Header (dghd)

Dataset Group Metadata (dgmd)

Dataset Group CDL Data (dgcd)

Dataset Group Protection (dgpr)

Dataset (dtn)

Dataset Header (dthd)

Dataset Metadata (dtmd)

Dataset CDL Metadata (dtn)

Dataset Protection (dtn)

Dataset Parameter Set (pars)

...

Annotation Table (atcn)

Annotation Table Header (athd)

Annotation Table Metadata (atmd)

Annotation Table CDL Data (atcd)

Annotation Table Protection (atpr)

Attribute Group Tile Configuration (agtc)

Attribute Data Tile Structure (adts)

...

Attribute Data Byte Offset (adbo)

Annotation Table Index (atix)

Genomic Range Index (grix)

...

Attribute Value Index (avix)

...

Attribute Group (agcn)

Attribute Group Header (aghd)

Annotation Access Unit (aauc)

Annotation Access Unit Header (aaah)

Annotation Access Unit Protection (aaup)

AAU Block

AAU Block

...

AAU Block

AAU Block

⋮

Annotation Access Unit (aauc)

Annotation Access Unit Header (aaah)

Annotation Access Unit Protection (aaup)

AAU Block

AAU Block

...

AAU Block

AAU Block

Figure 6 — Data structures hierarchy for storage of genomic annotation data

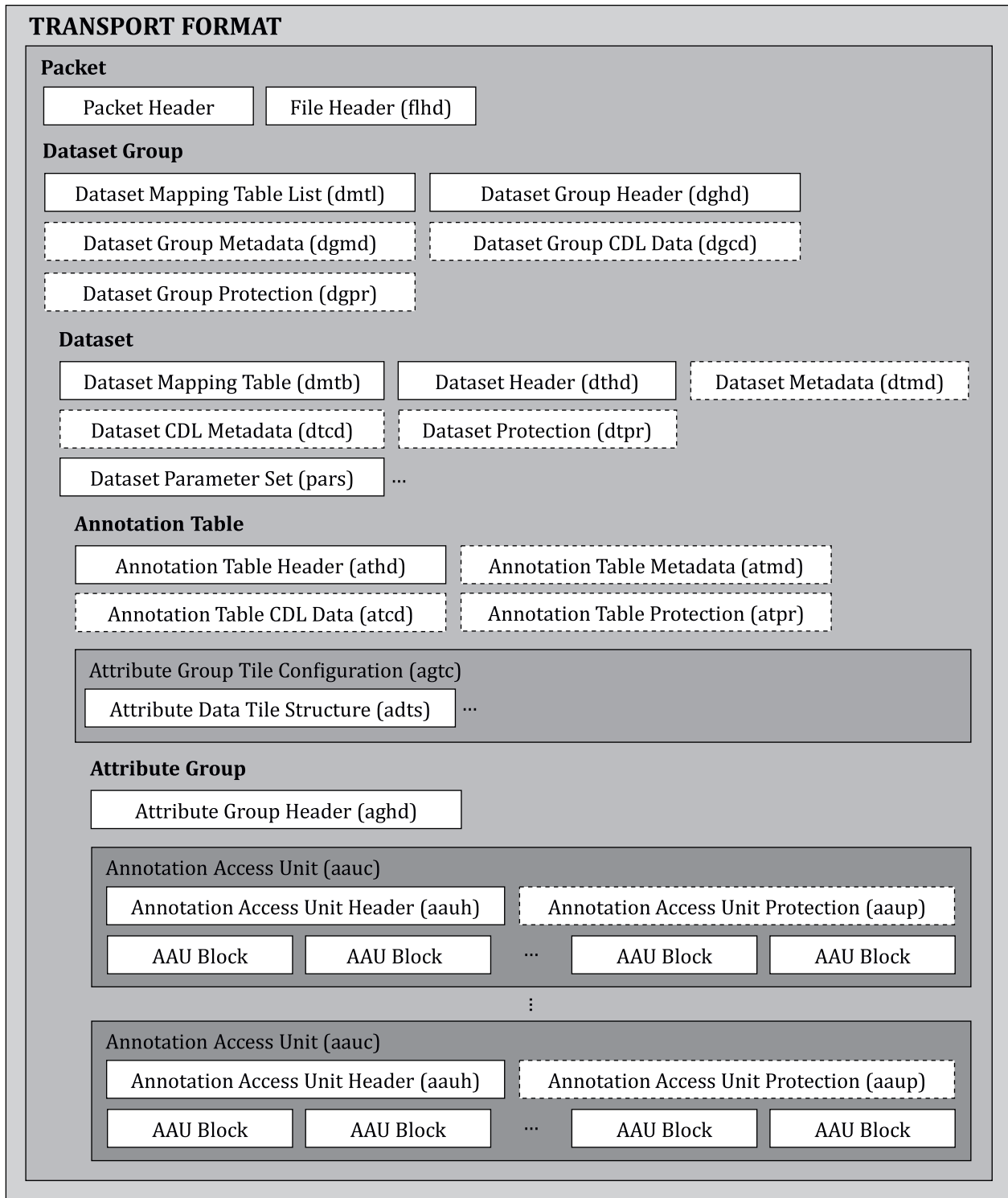


Figure 7 — Data structures hierarchy for transport of genomic annotation data

In transport format, any box represented in [Figure 5](#) and [Figure 7](#) shall be encapsulated in one or more packets, as specified in [subclause 6.6.5](#). The depacketization process is illustrated in [Annex D](#). The dataset group, dataset, annotation table and attribute group are represented in [Figure 5](#) and [Figure 7](#) for clarity, but the corresponding container boxes (dgc, dcn, atcn and agcn) 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 for genomic sequencing data, and
 - dataset group, dataset, annotation table, attribute group tile configuration, annotation table index, attribute group and annotation access unit for genomic annotation data shall be ordered according to the hierarchy specified in [Table 10](#).
- 2) The box order inside the containers dgcn, dtcn, dscn, aucn, atcn, agtc, atix, agcn and aauc is specified in [Table 13](#), [Table 25](#), [Table 63](#), [Table 33](#), [Table 40](#), [Table 47](#), [Table 59](#), [Table 49](#) and [Table 52](#) 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 6.5.4](#), 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 6.5.4](#)

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 10](#).
- 2) It is recommended to transmit the boxes in hierarchical order, as specified in [Table 10](#).
- 3) It is recommended, when possible, within each dataset group, to interleave boxes with hierarchical level equal to 1 (second column in [Table 10](#)), and the dthd and pars boxes belonging to different datasets, before transmitting all other boxes in the corresponding datasets.
- 4) For genomic annotation data, it is recommended, when possible, within each dataset, to interleave boxes with hierarchical level equal to 2 (third column in [Table 10](#)), and the athd, agtc and aghd boxes belonging to different annotation tables, before transmitting all other boxes in the corresponding annotation tables.

6.2 Syntax for representation

KLV (Key Length Value) format is used for all the data structures listed in [Table 10](#) 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 6.5.4](#).

All syntax tables specified in [subclauses 6.4](#), [6.5](#), and [6.6](#) 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.

6.3 Output data unit

This subclause specifies the output data unit (see [Table 11](#)) of the decapsulation processes specified in [subclauses 6.4.2.3](#), [6.4.3.7](#), [6.4.4.2](#), and [6.4.8.2](#).

Table 11 — Data unit syntax

Syntax	Type	Remarks
<code>data_unit() {</code>		
data_unit_type	<code>u(8)</code>	
<code>if (data_unit_type == 0) {</code>		
data_unit_size	<code>u(64)</code>	
<code>raw_reference()</code>		As specified in ISO/IEC 23092-2
<code>}</code>		
<code>else if (data_unit_type == 1) {</code>		
reserved	<code>u(10)</code>	
data_unit_size	<code>u(22)</code>	
<code>parameter_set()</code>		As specified in ISO/IEC 23092-2
<code>}</code>		
<code>else if (data_unit_type == 2){</code>		
reserved	<code>u(3)</code>	
data_unit_size	<code>u(29)</code>	
<code>access_unit()</code>		As specified in ISO/IEC 23092-2
<code>}</code>		
<code>else if (data_unit_type == 3){</code>		
reserved	<code>u(10)</code>	
data_unit_size	<code>u(22)</code>	
<code>annotation_parameter_set()</code>		As specified in ISO/IEC 23092-6
<code>}</code>		
<code>else if (data_unit_type == 4){</code>		
reserved	<code>u(3)</code>	
data_unit_size	<code>u(29)</code>	
<code>annotation_access_unit()</code>		As specified in ISO/IEC 23092-6
<code>}</code>		
<code>else /*(data_unit_type > 4)*/{</code>		
<code>/*skip data unit*/</code>		
<code>}</code>		
<code>}</code>		

`data_unit_type` and `data_unit_size` shall be filled as specified in [subclauses 6.4.2.3.6](#), [6.4.3.7.1](#), [6.4.4.1](#), and [6.4.8.1](#).

6.4 Data structures common to file format and transport format

6.4.1 File header

6.4.1.1 General

This box (*flhd* in [Table 12](#)) is mandatory and provides information about the version of this specification and of the set of other specifications the file or stream complies with.

6.4.1.2 Syntax

Table 12 — File header syntax

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

6.4.1.3 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:
 $\text{num_compatible_brands} = (\text{Length} - 22) / 4$.

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

6.4.2 Dataset group

6.4.2.1 General

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

The relevant container box (*dgcn* in [Table 13](#)) is mandatory in file format, forbidden in transport format.

Child boxes may be present or not, according to the column *Mandatory* in [Table 10](#). Child boxes marked with suffix “[]” after their name in the Syntax column of [Table 13](#) may be present in multiple instances.

Table 13 — Dataset group syntax

Syntax	Key	Type	Remarks
<i>dataset_group</i> {	<i>dgcn</i>		
dataset_group_header	dghd	gen_info	As specified in subclause 6.4.2.2
reference[]	rfgn	gen_info	As specified in subclause 6.4.2.3
reference_metadata[]	rfmd	gen_info	As specified in subclause 6.4.2.4
label_list	labl	gen_info	As specified in subclause 6.4.2.5
DG_metadata	dgmd	gen_info	As specified in subclause 6.4.2.6
DG_CDL	dgcd	gen_info	As specified in subclause 6.4.2.7
DG_protection	dgpr	gen_info	As specified in subclause 6.4.2.8
for (i=0;i<num_datasets;i++) {			num_datasets: as specified in subclause 6.6.3.2
dataset[i]	dtcn	gen_info	As specified in subclause 6.4.3.1
}			
}			

6.4.2.2 Dataset group header

6.4.2.2.1 General

This is a mandatory box (*dghd* in [Table 14](#)) describing the content of a dataset group.

6.4.2.2.2 Syntax

Table 14 — Dataset group header syntax

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

6.4.2.2.3 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[i] 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.

6.4.2.3 Reference

6.4.2.3.1 General

This is an optional box (*rfgn* in [Table 15](#)) 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.

6.4.2.3.2 Syntax

Table 15 — Reference box syntax

Syntax	Key	Type	Remarks
<i>reference</i> {	<i>rfgn</i>		
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 6.4.1.3
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 6.4.2.3.4
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 6.4.1.3
ref_checksum		u(checksum_size)	As specified in 6.4.2.3.7
}			

Table 15 (continued)

Syntax	Key	Type	Remarks
if (minor_version != '1900' reference_type != MPEGG_REF) {			minor_version as specified in subclause 6.4.1.3
for(seqID=0;seqID<seq_count;seqID++) {			
ref_seq_checksum [seqID]		u(checksum_ size)	As specified in subclause 6.4.2.3.7
}			
}			
}			
else {			
internal_dataset_group_ID		u(8)	
internal_dataset_ID		u(16)	
}			
}			

6.4.2.3.3 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 6.4.2.2](#).

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[seqID] is an unambiguous string identifier for each reference sequence contained in the reference.

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

sequence_ID[seqID] 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 6.4.3](#), 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 6.4.2.3.4](#).

reference_type specifies the type of the external reference and can take any of the values in the first column of [Table 16](#).

Table 16 — reference_type values

Value	Name	Semantics
0	MPEG_REF	Reference encoded as a dataset, as specified in subclause 6.4.3 , 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 6.4.3.2 , 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 6.4.2.3.5
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 6.4.2.3.7](#), on the payload of the dataset of type 2 (Value field of dtcn box), as specified in [subclause 6.4.3](#), retrieved using ref_uri, external_dataset_group_ID and external_dataset_ID.

ref_seq_checksum[seqID] is the checksum computed, according to one of the methods specified in [subclause 6.4.2.3.7](#), 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 6.4.2.3.6](#), if reference_type is equal to FASTA_REF or MPEG_REF.

internal_dataset_group_ID is an integer number identifying the dataset group containing the internal reference. An internal reference shall be of type MPEG_REF, as specified in [Table 16](#).

internal_dataset_ID is an integer number identifying the dataset containing the internal reference. An internal reference shall be of type MPEG_REF, as specified in [Table 16](#).

6.4.2.3.4 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](#).

6.4.2.3.5 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.

[Table 17](#) is an example of the supported FASTA format.

Table 17 — Example of the FASTA format

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	
...	...	
...	...	...

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

6.4.2.3.6 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 16](#), the corresponding dataset shall be decapsulated, according to [subclause 6.4.3](#), 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 16](#), 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 16](#), the FASTA reference, as specified in [subclause 6.4.2.3.6](#), 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 6.4.2.3.2](#).

In all of the above three cases the output raw reference shall be encapsulated as payload of a data unit, as specified in [subclause 6.3](#), 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.

6.4.2.3.7 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 6.4.2.3.2](#). Two values of `checksum_alg` are defined in [Table 18](#), while other values are reserved for future use.

Table 18 — 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.

6.4.2.4 Reference metadata

6.4.2.4.1 General

This is an optional box (*rfmd* in [Table 19](#)) containing metadata associated to a reference.

6.4.2.4.2 Syntax

Table 19 — Reference metadata syntax

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

6.4.2.4.3 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 6.4.2.3](#), present in the dataset group.

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

6.4.2.5 Label List

6.4.2.5.1 General

This box (*labl* in [Table 20](#)) lists the labels, as specified in [subclause 6.4.2.5.2](#), associated to a dataset group.

6.4.2.5.2 Syntax

Table 20 — Label list syntax

Syntax	Key	Type	Remarks
<i>label_list</i> {	<i>lbl</i>		
dataset_group_ID		u(8)	
num_labels		u(16)	
for (h=0; h < num_labels; h++) {			
label[h]	<i>lbl</i>	gen_info	As specified in subclause 6.4.2.5.4
}			
}			

6.4.2.5.3 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 6.4.2.2](#).

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

6.4.2.5.4 Label

6.4.2.5.4.1 General

A label (*lbl* in [Table 21](#)) is an identifier associated to one or more datasets, genomic regions and/or classes.

6.4.2.5.4.2 Syntax

Table 21 — Label syntax

Syntax	Key	Type	Remarks
<i>label</i> {	<i>lbl</i>		
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)	
}			
}			

Table 21 (continued)

Syntax	Key	Type	Remarks
<code>while(!byte_aligned())</code>			As specified in subclause 4.2.3
<code>nesting_zero_bit</code>		f(1)	Equal to 0
<code>}</code>			

6.4.2.5.4.3 Semantics

label_ID is a string representing the label identifier in the label list specified in [subclause 6.4.2.5](#). The variable stringLength for this field, as specified in [subclause 4.2.3](#), 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[i] 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 6.4.2.2](#).

num_regions[i] is the number of regions labelled by label_ID in the dataset.

seq_ID[i][j] 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 6.4.2.3](#).

num_classes[i][j] is the number of classes labelled by label_ID in the region.

class_ID[i][j][k] identifies the data class in the region labelled by label_ID, as specified in [Table 6](#).

start_pos[i][j] 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[i][j] 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.

6.4.2.6 Dataset group metadata

6.4.2.6.1 General

This is an optional box (*dgmd* in [Table 22](#)) containing metadata associated to a dataset group.

6.4.2.6.2 Syntax

Table 22 — Dataset group metadata syntax

Syntax	Key	Type	Remarks
<code>DG_metadata {</code>	<i>dgmd</i>		
<code>if (minor_version != '1900') {</code>			minor_version as specified in subclause 6.4.1.3
<code>dataset_group_ID</code>		u(8)	
<code>}</code>			
<code>DG_metadata_value()</code>			As specified in ISO/IEC 23092-3
<code>}</code>			

6.4.2.6.3 Semantics

dataset_group_ID is the identifier of the dataset group containing this metadata.

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

6.4.2.7 Dataset group clinical data linkage metadata

6.4.2.7.1 General

This is an optional box (*dgcd* in [Table 23](#)) containing clinical data linkage metadata associated to a dataset group.

6.4.2.7.2 Syntax

Table 23 — Dataset group clinical data linkage metadata syntax

Syntax	Key	Type	Remarks
<i>DG_CDL</i> {	<i>dgcd</i>		
dataset_group_ID		u(8)	
<i>DG_CDL_value()</i>			As specified in ISO/IEC 23092-3
}			

6.4.2.7.3 Semantics

dataset_group_ID is the identifier of the dataset group containing this clinical data linkage metadata.

DG_CDL_value() contains the dataset group clinical data linkage metadata, as specified in ISO/IEC 23092-3.

6.4.2.8 Dataset group protection

6.4.2.8.1 General

This is an optional box (*dgpr* in [Table 24](#)) 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.

6.4.2.8.2 Syntax

Table 24 — Dataset group protection syntax

Syntax	Key	Type	Remarks
<i>DG_protection</i> {	<i>dgpr</i>		
if (minor_version != '1900') {			minor_version as specified in subclause 6.4.1.3
dataset_group_ID		u(8)	
}			
<i>DG_protection_value()</i>			As specified in ISO/IEC 23092-3
}			

6.4.2.8.3 Semantics

dataset_group_ID is the identifier of the dataset group containing this protection metadata.

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

6.4.3 Dataset

6.4.3.1 General

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

The corresponding container box (*dtn* in [Table 25](#)) is mandatory in file format, forbidden in transport format.

Child boxes may be present or not, according to the column “Mandatory” in [Table 10](#). Child boxes marked with suffix “[]” after their name in the Syntax column of [Table 25](#) may be present in multiple instances.

Table 25 — Dataset syntax

Syntax	Key	Type	Remarks
<i>dataset</i> {	<i>dtn</i>		
dataset_header	dthd	gen_info	As specified in subclause 6.4.3.2
DT_metadata	dtmd	gen_info	As specified in subclause 6.4.3.3
DT_metrics	dtmt	gen_info	As specified in subclause 6.4.3.4
DT_CDL	dtcd	gen_info	As specified in subclause 6.4.3.5
DT_protection	dtpr	gen_info	As specified in subclause 6.4.3.6
dataset_parameter_set[]	pars	gen_info	As specified in subclause 6.4.3.7
if (dataset_type < 3) {			As specified in subclause 6.4.3.2
if (MIT_flag == 1) {			As specified in subclause 6.4.3.2
master_index_table	mitb	gen_info	As specified in subclause 6.5.2.1
}			
access_unit[]	aucn	gen_info	As specified in subclause 6.4.4.1
if (block_header_flag == 0) {			
descriptor_stream[]	dscn	gen_info	As specified in subclause 6.5.3.1
}			
}			
else if (dataset_type == 3) {			
annotation_table[]	atcn	gen_info	As specified in subclause 6.4.6.1
}			
}			

6.4.3.2 Dataset header

6.4.3.2.1 General

This is a mandatory box (*dthd* in [Table 26](#)) describing the content of a dataset.

6.4.3.2.2 Syntax

Table 26 — Dataset header syntax

Syntax	Key	Type	Remarks
<i>dataset_header</i> {	<i>dthd</i>		
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)	
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) {			

Table 26 (continued)

Syntax	Key	Type	Remarks
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
}			
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] */			
}			
}			
if(dataset_type == 3){			
n_ann_tables		u(7)	
for (i = 0; i < n_ann_tables; i++) {			
AT_ID [i]		u(8)	
AT_index_exists [i]		u(1)	
}			
}			
else if (dataset_type > 3){			
/* reserved for future use */			
}			
while(!byte_aligned())			As specified in subclause 4.2.3
nesting_zero_bit		f(1)	Equal to 0
}			

6.4.3.2.3 Semantics

dataset_group_ID is the identifier of the 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 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 6.5.2.1](#), 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 [subclauses 6.4.4.3.2](#), [6.5.2.1.2](#) and [6.5.2.4.4.2](#) 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 6.4.5.2](#). It is always set to 1 in transport format. See also [subclause 6.4.3.2.5](#).

MIT_flag: if set, the master index table, as specified in [subclause 6.5.2.1](#) 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 7](#), 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 6.4.3.2.5](#).

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 6.5.2.1](#). See also [subclause 6.4.3.2.5](#).

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 6.4.2.3.1](#), 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 6.4.2.3](#). 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 6.4.2.3](#), is equal to 0, then the reference box pointed by reference_id, as specified in [subclause 6.4.2.3](#), 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[seq] 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 6.4.2.3](#).

seq_blocks[seq] is the number of access units per reference sequence with a different value of access_unit_ID, as specified in [subclause 6.4.4.3](#). 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; 3 = annotation.

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 6](#). It shall take any of the values defined as Class ID in [Table 6](#). 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[ci] is the maximum number of descriptors per class encoded in the dataset.

descriptor_ID[ci][di] 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 6.4.3.7.2](#), 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 6.4.3.7.3](#), 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 6.4.3.7.3](#), 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 6.4.3.7.3](#), 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 6.4.3.7.3](#), 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 6.4.4.3](#), such fields will take the value they have in the dataset parameter set, as specified in [subclause 6.4.3.7](#), referenced by the access unit via the parameter_set_ID field, as specified in [subclause 6.4.4.3](#)
- when used in the master index table, as specified in [subclause 6.5.2.1](#), 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 27](#) in [subclause 6.4.3.2.4](#).

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 26](#).

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

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

n_ann_tables is the number of annotation tables available in the dataset when dataset type is set to 3.

AT_ID[] shall correspond to the value of the AT_ID variable in one of the annotation tables contained in the dataset, as specified in [subclause 6.4.6.2](#).

AT_index_exists[i] is a flag, if set to 1, indicates the presence of index data for the annotation table identified by AT_ID[i].

6.4.3.2.4 Alphabets

Each Alphabet is identified by an alphabet_ID as shown in [Table 27](#):

Table 27 — alphabet_ID semantics

alphabet_ID	$S_{\text{alphabet_ID}}$	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 $S_{\text{alphabet_ID}}[\text{index}]$ 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.

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

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

6.4.3.2.5 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 6.4.3.2.5](#), belonging to the same descriptor stream, as specified in [subclause 6.5.3](#), 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 6.4.5](#), belonging to the same access unit, as specified in [subclause 6.4.4](#), 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 6.4.4.3](#), or master index table, as specified in [subclause 6.5.2.1](#)). 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 $((1 \ll \text{byteOffsetSize}) - 1)$). 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 $((1 \ll \text{byteOffsetSize}) - 1)$, the smallest value greater than the offset of the block, if any (otherwise the descriptor_stream_size should be used as above).