FINAL DRAFT
International
Standard

ISO/IEC
FDIS
23092-3

Information technology — Genomic information representation —

Part 3: Metadata and application programming interfaces (APIs)

Technologie de l'information — Représentation des informations génomiques —

Partie 3: Métadonnées et interfaces de programmation d'application (API)

ISO/IEC JTC 1/SC 29

Secretariat: **JISC**

Voting begins on:
2025-03-05

Voting terminates on:
2025-04-30

Reference number
ISO/IEC FDIS 23092-3:2025(en)

Contents

Page

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 23092-3:2022), which has been technically revised.

The main changes are as follows:

— addition of annotation table metadata (subclause 6.5) that contains general, analytics, linkages and access history information associated with an annotation table;

— addition of metrics metadata (Clause 7) that contains pre-computed sequencing data metrics associated with a dataset or an access unit;

— addition of clinical data linkage metadata (Clause 8) that contains linkage information for enabling clinical data interchange (CDI) with external data sources;

— addition of annotation table protection metadata, including encryption parameters (subclause 9.2.4) and digital signatures (subclause 9.4.4), and updates to the decryption process (subclause 9.2.6) and privacy rules (subclause 9.3) for enabling the selective protection of annotation data;

— extension of the APIs (Clause 12) for supporting the random access and query of annotation data, the retrieval of pre-computed sequencing data statistics, and the return of only the number of matching records without the actual data.

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 the clinic. 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 the addressing of one limitation of the textual SAM format, for which incremental ad-hoc addition of features followed along the years, resulting in an overall redundant and suboptimal format which at the same time results not general and unnecessarily complicated;

— normatively separating free-field user-defined information with no clear semantics from the normative 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.

This document 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 normative procedure to assess conformance to the standard 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 specification.

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 confirming to 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 to develop 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.

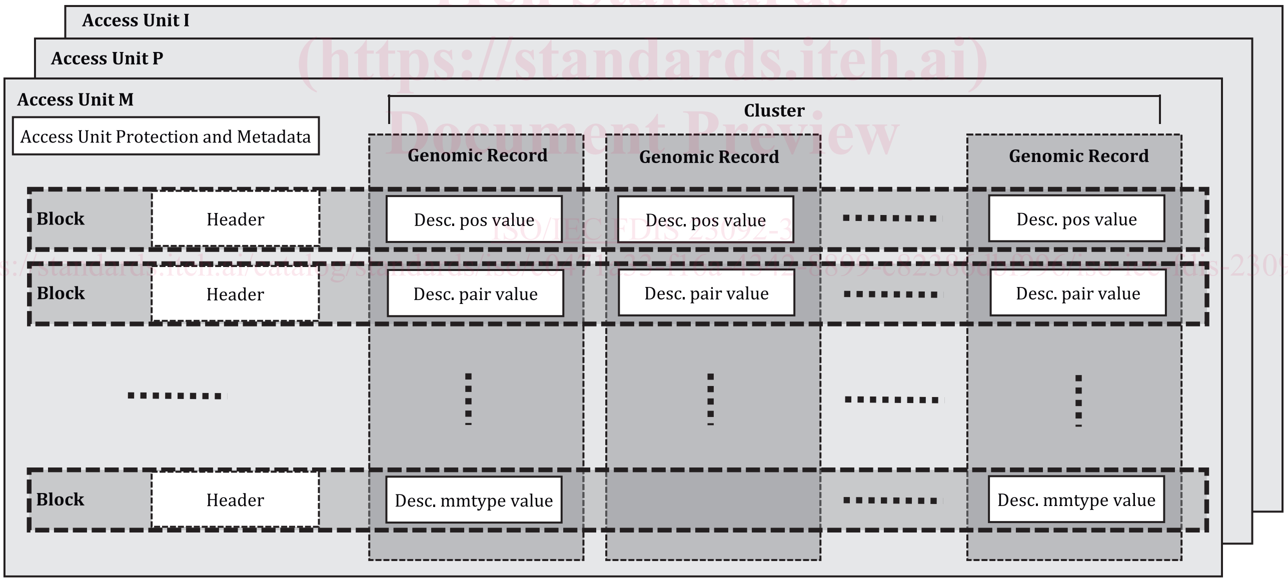

Figure 1 — Access units, blocks and genomic records

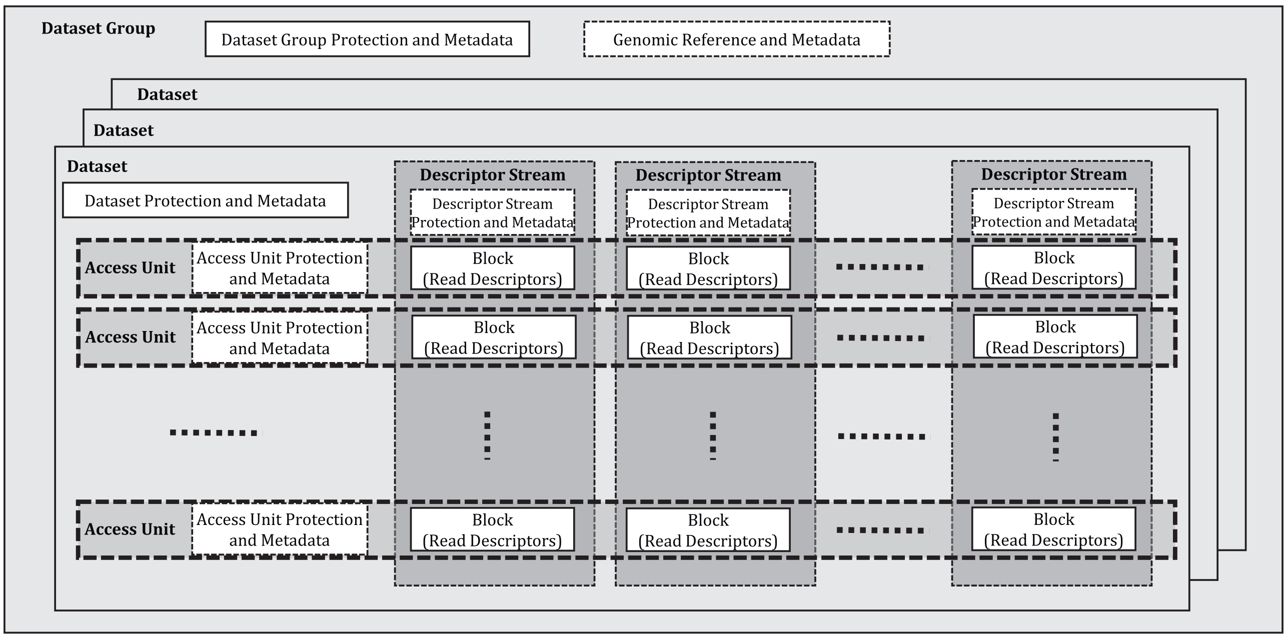
Dataset Group
Dataset Group Protection and Metadata
Genomic Reference and Metadata
Dataset
Dataset
Dataset
Dataset Protection and Metadata
Descriptor Stream
Descriptor Stream Protection and Metadata
Access Unit
Access Unit Protection and Metadata
Block (Read Descriptors)

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 can, for example, contain the complete sequencing of an individual, or a portion of it. Other datasets can 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.

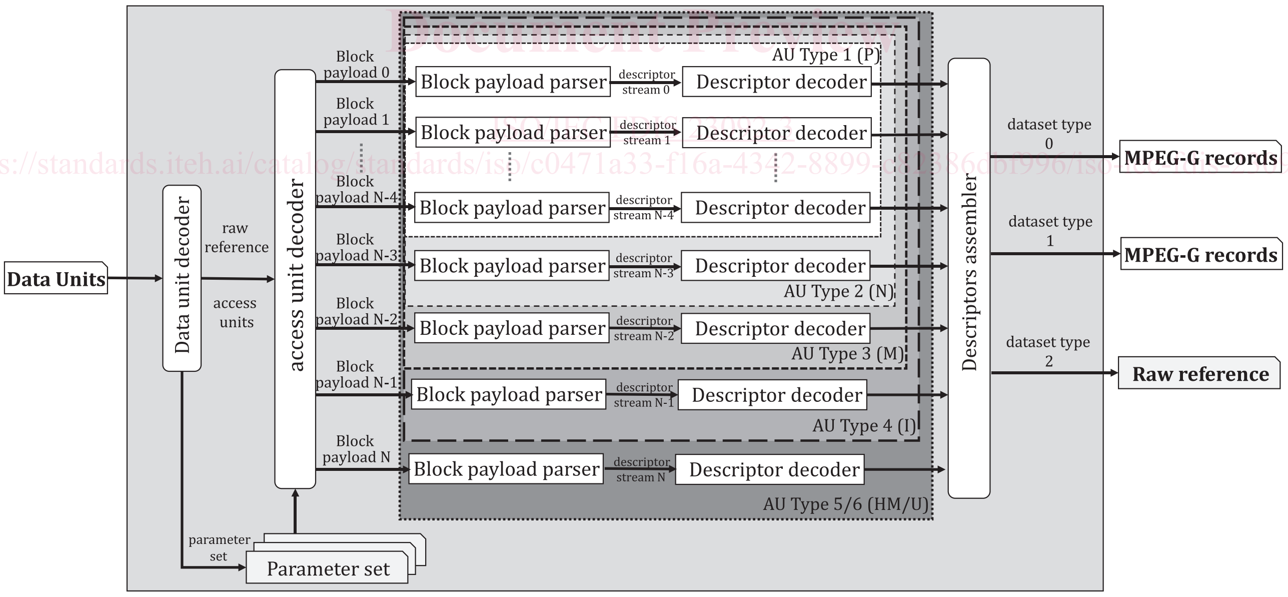

Figure 3 — Decoding process

Information technology — Genomic information representation —

Part 3: Metadata and application programming interfaces (APIs)

1 Scope

This document specifies information metadata, metrics metadata, clinical data linkage metadata, auxiliary fields, SAM interoperability, protection metadata and programming interfaces of genomic information. It defines:

— metadata storage and interpretation for the different encapsulation levels as specified in ISO/IEC 23092-1 (in Clause 6);

— metrics metadata containing sequencing data metrics at the dataset and access unit levels as specified in ISO/IEC 23092-1 (in Clause 7);

— clinical data linkage metadata stored at the dataset group, dataset and annotation table levels as specified in ISO/IEC 23092-1 (in Clause 8);

— protection elements providing confidentiality, integrity and privacy rules at the different encapsulation levels as specified in ISO/IEC 23092-1 (in Clause 9);

— how to associate auxiliary fields to encoded reads (in Clause 10);

— interfaces to access genomic information coded in compliance with ISO/IEC 23092-1 and ISO/IEC 23092-2 (in Clause 12);

— mechanisms for backward compatibility with existing SAM content, and exportation to this format (in Annex E).

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 23092-1, *Information technology — Genomic information representation — Part 1: Transport and storage of genomic information*

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

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

OASIS. eXtensible Access Control Markup Language (XACML) Version 3.0, 2013, Available: http://docs.oasis-open.org/xacml/3.0/xacml-3.0-core-spec-cs-01-en.pdf

W3C, XML Path Language (XPath), Version 1.0, 16 November 1999, Available: https://www.w3.org/TR/xpath-10/

IEEE. 754-2008, IEEE Standard for Floating-Point Arithmetic, August 2008, Available: https://ieeexplore.ieee.org/document/4610935

3 Terms and definitions

For the purposes of this document, the terms and definitions in ISO/IEC 23092-1, ISO/IEC 23092-2 and the following 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
BAM
compressed binary version of SAM

3.2
dataset group
collection of one or more datasets

Note 1 to entry: Which information is represented varies depending on the genomic information representation.

4 Abbreviated terms

AAU	annotation access unit
AU	access unit
AUC	access unit contiguity
DSC	descriptor stream contiguity
EBI	European Bioinformatics Institute
EGA	European Genome Archive
ENA	European Nucleotide Archive
LSB	least significant bit
NCBI	National Center for Biotechnology Information
SRA	sequence read archive
URN	uniform resource name

5 Conventions

5.1 Character encoding

The implementation of the specifications described in this document shall use UTF-8 character encoding.

5.2 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).

5.3 Syntax functions and data types

The functions presented here are used in the syntactical description. These functions are expressed in terms of the value of a bitstream pointer that indicates the position of the next bit to be read by the decoding process from the bitstream.

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.

Size(array_name[]) returns the number of elements contained in the array named array_name.

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).

— 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) begins at a byte-aligned position in the bitstream and 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-aligned 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.

  NOTE The st(v) syntax data type is only used in this document when the current position in the bitstream is a byte-aligned position.

— i(n): signed 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 two's complement integer representation with most significant bit written first.

— 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.

— F32: 32 bit single precision floating-point as specified by IEEE 754-2008. The parsing process is specified as follows: u(1) is used for the sign value, followed by an u(8) used for the exponent value, followed by an u(23) used for the fraction value.

— F64: 64 bit double precision floating-point as specified by IEEE 754-2008. The parsing process is specified as follows: u(1) is used for the sign value, followed by an u(11) used for the exponent value, followed by an u(52) used for the fraction value.

— c(n): sequence of n ASCII characters.

5.4 Graphic notations

The notation `->` (arrow) is used in this document to indicate the access to a member of a data structure.

The notations `|` `|=` are used in this document to indicate the bitwise OR operation and assignment, respectively. `a |= b` is equivalent to `a = a | b`.

The notations `&` `&=` are used in this document to indicate the bitwise AND operation and assignment, respectively. `a &= b` is equivalent to `a = a & b`.

The notation `return_error()` is used in this document to indicate that the decoding process has to stop due to a decoding error which cannot be handled.

The notation `continue` is used in this document within `for` and `while` statements to signal that the process shall continue to the next iteration without executing any further statement in the current iteration.

The notation `*(ptr)` is used in this document to access the data/value in the memory that the pointer `ptr` points to - the contents of the address with that numerical index. The operator * is said to *dereference* the pointer `ptr`.

The notation `strncmp(str1, str2, n)` is used in this document to indicate if first n characters of two strings `str1` and `str2` match with each other. If all `n` characters match, then it returns 0, otherwise 1.

6 Information metadata

6.1 General

This clause defines a minimum core set of metadata elements, which users and applications can then extend by including extra information elements. Metadata sets are specified for dataset groups, datasets, references and annotation tables, as specified in ISO/IEC 23092-1. The structure of these metadata sets and their elements is specified using XML v1.1.

Extensions to (i.e. new elements for) the metadata sets specified in this clause are represented with an identifier of the extension type in the form of a URI, a value and a pointer to a resource documenting the semantics of the extension type.

Metadata profiles are specific subsets of metadata sets specified using mechanisms provided in this document. A metadata profile specified in this document may correspond to well-known metadata sets specified or used out of the ISO/IEC 23092 series, such as those in ENA or EGA[1] and NCBI specifications.[2] This allows easy interoperability with already existing systems. A metadata profile includes a subset of (or all) core elements described in subclauses 6.2 and 6.4, and a set of new elements specified with the extension mechanism specified in subclause 6.7.

The rest of clauses specify dataset group metadata (see subclause 6.2), reference metadata (see subclause 6.3), dataset metadata (see subclause 6.4), annotation table metadata (see subclause 6.5), extensions (see subclause 6.7) and profiles (see subclause 6.8).

6.2 Dataset group metadata

Compressed dataset group metadata are stored within the `DG_metadata_value` element of the *DG_metadata* box (with key `dgmd`), as specified in ISO/IEC 23092-1. The decoding process of `DG_metadata_value` is specified in Clause 11. The output of the decoding process is an XML document, where the root node is `DatasetGroupMetadata`. Annex A.1 provides the XML schema for a decoded dataset group metadata.

As previously introduced in subclause 6.1, an extensions type is the combination of three elements: the value, the identifier of the extension, and a link to a resource documenting the interpretation of the extenstion. In the XML schema, this is translated as an element with three child elements: the `Type` element (of type URI), the `Documentation` (of type URI) and the value which is represented as the element taking the place of the element `any` in the schema. Additionally, for extensions belonging to the dataset group, the Boolean element `Inheritable` (as specified in Annex A.1) of the extension element indicates if the extension is only relevant to the dataset group, or if the dataset also inherits it. The resource documentation can be human readable, and the extensions parsing is not required.

6.3 Reference metadata

Compressed reference metadata are stored within the reference metadata box, as specified in ISO/IEC 23092-1, in the `reference_metadata_value` field. Clause 11 specifies the decoding process of `reference_metadata_value`. The output of the decoding process is an XML document, with a root element `ReferenceMetadata`. Annex A.7 provides the related XML schema. Table 1 specifies the semantics of the fields.

Table 1 — Semantics of reference sequence's fields

Tag name	Description
length	Length in base pairs[a] of the sequence
alternative_locus_location	The sequence is an alternative locus from an unknown region. A child element `chromosome_name` identifies on which chromosome the sequence has an alternative locus. If present, a child element `position` indicates the start and end position of the alternative locus.
alternative_sequence_name	List of alternative names
genome_assembly_identifier	Genome assembly identifier
description	Human readable textual description
species	Name of the species
URI	URI of the sequence

[a] In this document, "base" or "base pair" is used as a synonym for "nucleotide".

6.4 Dataset metadata

Compressed dataset metadata are stored within the `DT_metadata_value` field of the `DT_metadata` box (marked as `dtmd`), as specified in ISO/IEC 23092-1. Clause 11 specifies the decoding process of `DT_metadata_value`. The output of the decoding process is an XML document with an element `DatasetMetadata` as root. Annex A.2 provides the XML schema for dataset metadata. A dataset metadata element overwrites the corresponding element whose values differ from the one indicated at the dataset group level (i.e. the new value in the dataset is a specialization of the value at the dataset group level).

Table 2 defines the process to obtain the dataset metadata with inherited elements, where the following notations are used.

— .has(): the function returns true if the element has a child element with an unqualified name equal to the parameter given, and false otherwise.

— .get(): the function returns the content of the child element with an unqualified name equal to the parameter given, as an array of characters.

— .getElement(): the function returns the content of the child element with an unqualified name equal to the parameter given.

— .getByIndex(): the function returns the content of the ith child element with an unqualified name equal to the first parameter given and i equal to the second parameter given, as an array of characters.

— .getEncoding(): the function returns the content of the element as an array of characters.

— .set(): the function sets the content of the child element with an unqualified name equal to the first parameter given, to the array of characters given as the second parameter.

— .add(): the function creates a new child element with an unqualified name equal to the first parameter given, and a content equal to the second parameter. The created element is appended to the content of the current element.

— .getNumber(): the function returns the number of child elements with an unqualified name equal to the parameter given.

Table 2 — Decoding process of dataset metadata

```
datasetMetadataWithInheritance = datasetMetadata
if (!datasetMetadata.has(“Type”)) {
  datasetMetadataWithInheritance.set(
    “Type”, datasetGroupMetadata.get(“Type”)
  )
}
if (!datasetMetadata.has(“Abstract”)) {
  datasetMetadataWithInheritance.set(
    “Abstract”, datasetGroupMetadata.get(“Abstract”)
  )
}
if (!datasetMetadata.has(“ProjectCentre”)) {
  datasetMetadataWithInheritance.set(
    “ProjectCentre”, datasetGroupMetadata.get(“projectCentre”)
  )
}
if (!datasetMetadata.has(“Description”)) {
  datasetMetadataWithInheritance.set(
    “Description”, datasetGroupMetadata.get(“Description”)
  )
}
if (!datasetMetadata.has(“Samples”)) {
  datasetMetadataWithInheritance.set(
    “Samples”, datasetGroupMetadata.get(“Samples”)
  )
}
extensions = datasetGroupMetadata.getElement(“Extensions”)
extensionsDataset = datasetMetadata.getElement(“Extensions”)
for (i = 0; i < extensions.getNumber(“Extension”); i++) {
  extension = extensions.getByIndex(“Extension”, i)
  typeExtension = extension.get(“Type”)
  if (extension.get(“Inheritable”) == “false”) {
    continue
  }
  found = false
  for (j = 0; j < extensionsDataset.getNumber(“Extension”); j++){
    extensionDataset = extensionsDataset.getByIndex(“Extension”, j)
    typeExtensionDataset = extensionDataset.get(“Type”)
    if (typeExtension == typeExtensionDataset) {
      found = true
      break
    }
  }
  if (!found) {
    extensionsDataset.add(“Extension”, extension.getEncoding())
  }
}
```

After executing the inheritance process, the extensions are ordered in the alphabetical order of their Type element.