
**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)*



STANDARDSISO.COM : Click to view the full PDF of ISO/IEC 23092-3:2020



COPYRIGHT PROTECTED DOCUMENT

© ISO/IEC 2020

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
Fax: +41 22 749 09 47
Email: copyright@iso.org
Website: www.iso.org

Published in Switzerland

Contents

| | Page |
|--------------------------------------------------------------------------|-----------|
| Foreword | v |
| Introduction | vi |
| 1 Scope | 1 |
| 2 Normative references | 1 |
| 3 Terms and definitions | 2 |
| 4 Abbreviated terms | 2 |
| 5 Conventions | 2 |
| 5.1 Character encoding | 2 |
| 5.2 Syntax functions and data types | 2 |
| 5.3 Graphic notations | 3 |
| 6 Information metadata | 4 |
| 6.1 General | 4 |
| 6.2 Dataset group metadata | 4 |
| 6.3 Reference metadata | 4 |
| 6.4 Dataset metadata | 5 |
| 6.5 Metadata protection | 7 |
| 6.6 Mechanism for extensions of the metadata set | 7 |
| 6.6.1 General | 7 |
| 6.6.2 Example for dataset metadata extensions | 8 |
| 6.6.3 Example for obfuscating labels | 8 |
| 6.6.4 Example for obfuscating sequences | 8 |
| 6.7 Metadata profiles | 8 |
| 6.7.1 General | 8 |
| 6.7.2 Example of metadata profile — Run | 8 |
| 6.7.3 Example of metadata profile — Genomic data commons | 9 |
| 7 Protection metadata | 10 |
| 7.1 General | 10 |
| 7.2 Encryption of <code>gen_info</code> elements and blocks | 10 |
| 7.2.1 General | 10 |
| 7.2.2 EncryptionParameters carried in dataset group protection | 10 |
| 7.2.3 EncryptionParameters carried in dataset protection | 12 |
| 7.2.4 Key retrieval | 15 |
| 7.2.5 Decryption | 16 |
| 7.3 Privacy rules for the use of the genomic information | 17 |
| 7.3.1 General | 17 |
| 7.3.2 Example of use of privacy rules | 18 |
| 7.4 Digital signature of <code>gen_info</code> elements and blocks | 19 |
| 7.4.1 General | 19 |
| 7.4.2 Signatures carried in dataset group protection | 19 |
| 7.4.3 Signatures carried in dataset protection | 19 |
| 7.4.4 Signatures carried in access unit protection | 21 |
| 7.4.5 Signatures carried in descriptor stream protection | 21 |
| 8 Access unit information | 22 |
| 8.1 General | 22 |
| 8.2 <code>genAuxRecord</code> | 22 |
| 8.3 <code>genAux</code> | 23 |
| 8.4 <code>genTag</code> | 23 |
| 9 Decoding process for metadata | 24 |
| 9.1 General | 24 |
| 9.2 Initialization of parameters | 26 |
| 9.2.1 General | 26 |

| | | |
|---------------------|----------------------------------------------------------------------------------------------------|-----------|
| 9.2.2 | Properties | 26 |
| 9.2.3 | Parameters | 26 |
| 9.2.4 | Constants | 27 |
| 9.2.5 | Process | 28 |
| 9.3 | Macros | 29 |
| 9.4 | Decoding process | 31 |
| 10 | Application programming interfaces (APIs) | 38 |
| 10.1 | General | 38 |
| 10.2 | Structure of the API | 38 |
| 10.3 | Detailed specification of the API | 38 |
| 10.3.1 | Data types | 38 |
| 10.3.2 | Return codes | 39 |
| 10.3.3 | Metadata fields | 39 |
| 10.3.4 | Output structures | 40 |
| 10.3.5 | Filters | 47 |
| 10.3.6 | Genomic information | 52 |
| 10.3.7 | Metadata | 56 |
| 10.3.8 | Protection | 58 |
| 10.3.9 | Reference | 59 |
| 10.3.10 | Statistics | 60 |
| Annex A | (normative) XML schemas corresponding to metadata information and protection elements | 63 |
| Annex B | (informative) XML schemas and XML-based data | 64 |
| Annex C | (informative) SAM interoperability | 77 |
| Annex D | (informative) Example of key transport | 84 |
| Bibliography | | 88 |

Foreword

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

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

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

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

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

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

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

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

Introduction

The advent of high-throughput sequencing (HTS) technologies has the potential to boost the adoption of genomic information in everyday practice, ranging from biological research to personalized genomic medicine in 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 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 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.

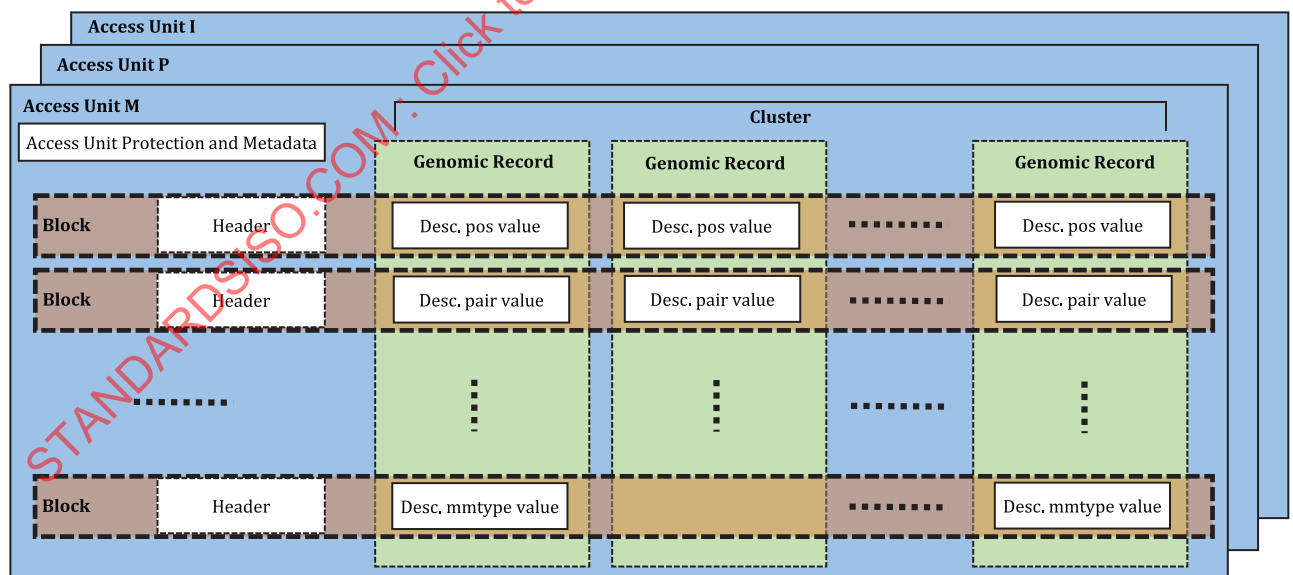


Figure 1 — Access units, blocks and genomic records

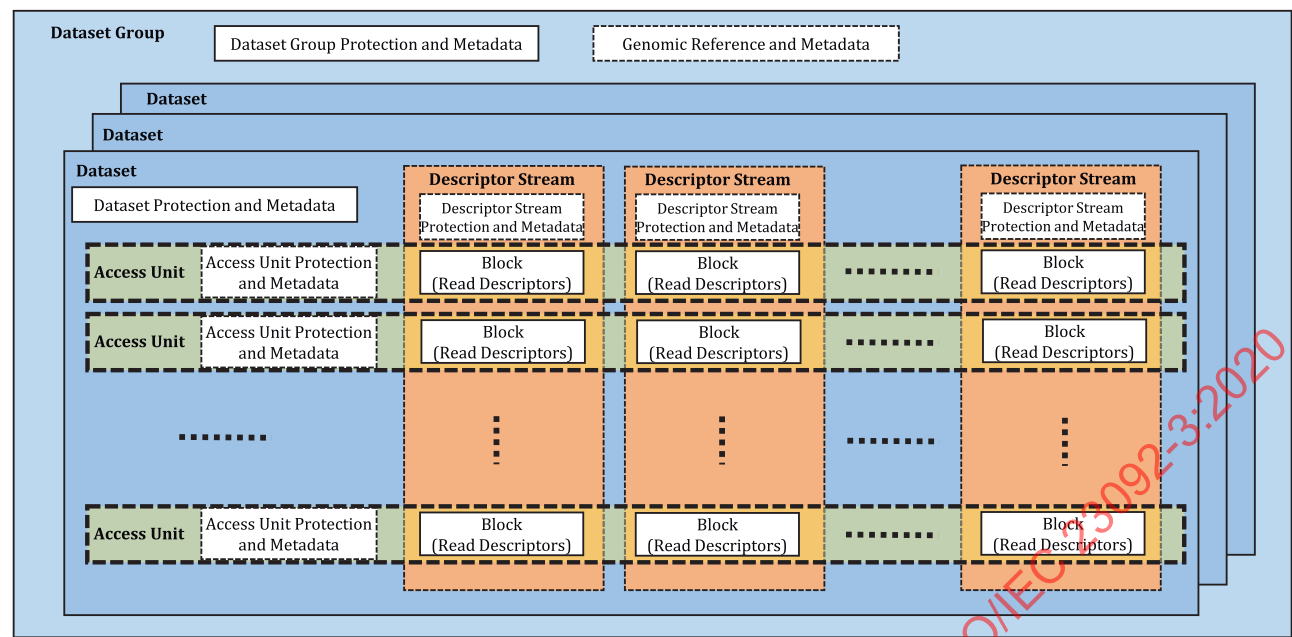


Figure 2 — High-level data structure: datasets and dataset group

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](#).

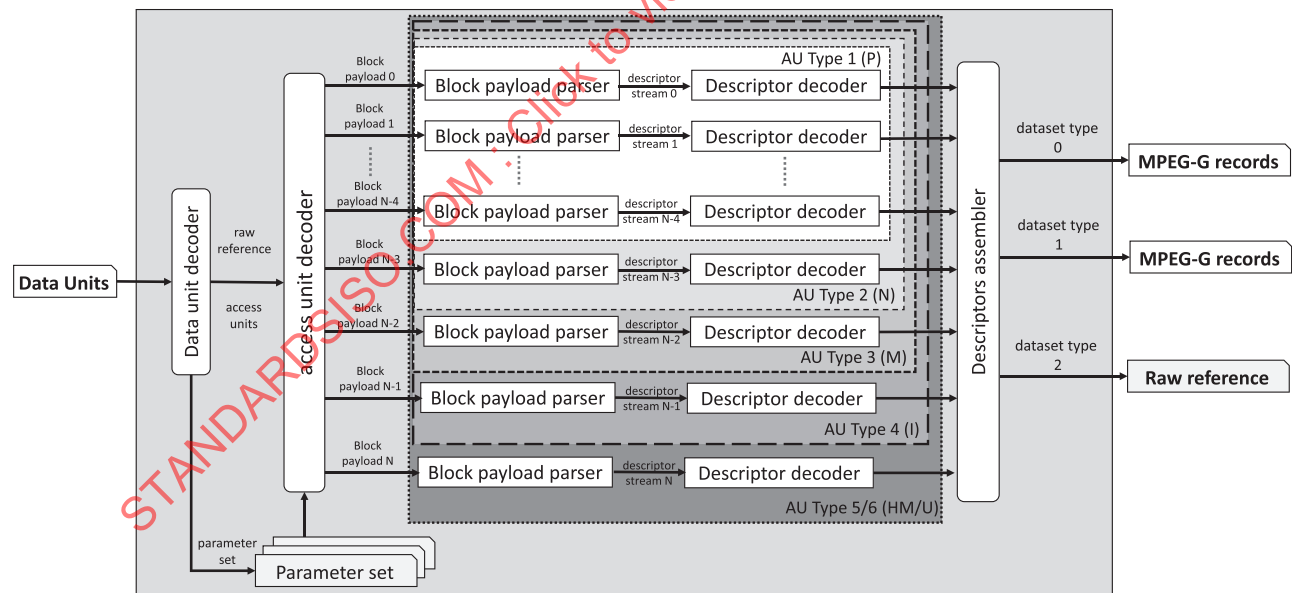


Figure 3 — Decoding process

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

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

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

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights other than those in the patent database. ISO and IEC shall not be held responsible for identifying any or all such patent rights.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC 23092-3:2020

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC 23092-3:2020

Information technology — Genomic information representation —

Part 3: Metadata and application programming interfaces (APIs)

1 Scope

This document specifies information 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](#));
- protection elements providing confidentiality, integrity and privacy rules at the different encapsulation levels specified in ISO/IEC 23092-1 (in [Clause 7](#));
- how to associate auxiliary fields to encoded reads (in [Clause 8](#));
- mechanisms for backward compatibility with existing SAM content, and exportation to this format (in [Annex C](#));
- interfaces to access genomic information coded in compliance with ISO/IEC 23092-1 and ISO/IEC 23092-2 (in subclause [8.1](#)).

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*

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>

IETF, PKCS #1: RSA Cryptography Specifications Version 2.2, November 2016, Available: <https://tools.ietf.org/html/rfc8017>

IETF, PKCS #5: Password-Based Cryptography Specification Version 2.1, January 2017, Available: <https://tools.ietf.org/html/rfc2898><https://tools.ietf.org/html/rfc8018>

IETF, Advanced Encryption Standard (AES) Key Wrap Algorithm, September 2002, Available: <https://tools.ietf.org/html/rfc3394>

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 and ISO/IEC 23092-2 and the following apply.

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

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

3.1 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

| | |
|------|-----------------------------------------------|
| 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 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 $(\text{stringLength} + 1) * 8$ 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.
- `f64`: 64 bit double precision floating-point as specified by IEEE 754-2008.
- `c(n)`: sequence of `n` ASCII characters.

5.3 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 .

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 and references, 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], as examples. 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 6.6.

The rest of clauses specify dataset group metadata (subclause 6.2), reference metadata (subclause 6.3), dataset metadata (subclause 6.4), extensions (subclause 6.6) and profiles (subclause 6.7).

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 9. The output of the decoding process is an XML document, where the root node is `DatasetGroup`. 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 extension. 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 9 specifies the decoding process of `reference_metadata_value`. The output of the decoding process is an XML document, with a root element `Reference`. Annex A.4 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 |
| ^a In this document, "base" or "base pair" is used as a synonym for "nucleotide". | |

Table 1 (continued)

| Tag name | Description |
|---------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| alternative_locus_location | The sequence is an alternative locus from an unknown region. A child element <code>chromosome_name</code> identifies on which chromosome the sequence has an alternative locus. If present, a child element <code>position</code> 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 9](#) specifies the decoding process of `DT_metadata_value`. The output of the decoding process is an XML document with an element `Dataset` 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. In this table, 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 i^{th} 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")
  )
}
```

Table 2 (continued)

```

}
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") == "true"){
        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.

For example, one can have datasets for patients A, B and C; therefore the dataset group metadata includes a list of samples representing A, B and C. The datasets then provide only one sample description (respectively of A, B or C). The base set of elements in the dataset metadata is the same as for the dataset group, but the elements are not mandatory (so there is no need to repeat them), since by default their values are considered equal to the values indicated in the dataset group. This is always the case for the values belonging to the core set, or by default for the extensions except for those cases that have the inheritance parameter (element named `Inheritable` in the definition of the extension type in Annex A.1) set to false.

As in the case of the dataset group metadata, Annex A.2 provides the resulting schema.

Also as in the case of the dataset group, the extension mechanism is available to include new attributes where necessary. See subclause 6.6.2 on extensions for an example in the case of dataset metadata.

6.5 Metadata protection

The schemas defined in Annexes A.1 and A.2 include choices to either provide certain values as plaintext or encrypted content. The encrypted content shall be such that after decryption, a metadata element is obtained which is valid according to the schemas, but which does not contain any encrypted information. The mechanism to transmit the knowledge of the keys shall be established through another channel.

No call to functions specified in 10.3.7 may return encrypted content.

The schemas defined in Annexes A.1 and A.2 allow for the signature of the metadata or parts thereof. No call to functions specified in 10.3.7 may return content for which the signature could not be verified.

6.6 Mechanism for extensions of the metadata set

6.6.1 General

This subclause provides a mechanism for adding new elements to the different core metadata sets (dataset group and dataset levels).

An extended element consists of:

- an information type identifier (provided in the form of a URI),
- a value,
- documentation (provided in the form of a URI).

In the case of extensions at the dataset group level, a third value, the inheritance flag of type Boolean, is optionally present. If not present, the default value is True. If the flag is set to True, the value of the extension is inherited by the datasets belonging to the dataset group. If the flag is set to False, the value only applies to the dataset group.

Annex A.1 defines the extension schema. Using extensions, the core metadata sets can be adapted to multiple use cases. This document defines profiles (see subclause 6.7), which rely on well-known extensions, defined in this document and for which the URI pointer is known. To be compliant with a profile specified in this document, a tool shall implement the list of extensions included in the profile.

At the end of the decoding process, the extensions have to be ordered in growing lexicographic order based on the information type identifier.

This subclause presents examples of datasets using the concept of label specified in ISO/IEC 23092-1.

6.6.2 Example for dataset metadata extensions

ISO/IEC 23092-1 introduces the concept of dataset `label`, which supports the identification of one or more genomic intervals and the related data classes with a single identifier. Labels are not conceived to provide detailed documentation of the identified genomic regions. [Table 3](#) shows how to associate an ontology term to a label.

Table 3 — Sample metadata extended with a `label` linked to an ontology

| Field name | Field type | Mandatory |
|---------------|------------|-----------|
| Label name | st(v) | Yes |
| Ontology term | URN | Yes |

6.6.3 Example for obfuscating labels

In case the label name is considered sensitive information to be protected, this document specifies ways to obfuscate label names. One solution is to provide an extension to the metadata, which provides a translation between the obfuscated name and the real name. Using XML encryption, the access to the sensitive information can be restricted. Annex [B.5](#) provides a schema for the extension supporting multiple labels obfuscation where a transmitted label name is translated to a non-obfuscated one.

6.6.4 Example for obfuscating sequences

In order to maintain privacy, it can be necessary to hide which genomic region an access unit maps to. In order to hide this piece of information, it is necessary to obfuscate the sequences. One solution is then to provide an extension to the metadata, which provides a translation between the obfuscated sequence name and the real name. Access to the sensitive information can then be restricted using XML encryption. Annex [B.5](#) provides a schema for the extension supporting multiple sequence obfuscation where a transmitted sequence name is translated to a non-obfuscated one, and the real sequence start position.

6.7 Metadata profiles

6.7.1 General

Profiles are specific metadata extension sets. Subclause [6.6](#) provides the mechanism to specify them. Subclause [6.7.2](#) provides a formalized profile.

A profile corresponds to a well-known metadata set. Profiles are specified in this document, such as the one defined to support the Run sets of the SRA (Sequence Read Archive) schema version 1.5^[1]. Additional profiles may be specified following the mechanisms defined in this document.

A profile includes a subset of (or all) the core elements described in this document, and a set of new elements specified with the extension mechanism (see subclause [6.6](#)).

The metadata schemas Annexes [A.1](#) and [A.2](#) define the attribute `profile` in the dataset group and dataset metadata XML element, to define the profile being used. The profile is identified with a URI. In case no profile is active, the attribute is not used.

6.7.2 Example of metadata profile — Run

The ISO/IEC 23092 series dataset metadata shares characteristics with both the concepts of run and analysis as used by EGA^[3]. This series profile provides interoperability with the existing metadata schemas, such that an ISO/IEC 23092 series dataset is interoperable with an EGA run element.

This profile is identified with the URI: “urn:mpeg:mpeg-g:metadata:profile:ega:run”.

To be compliant with the profile, the following extension shall be present at the dataset group metadata level

- One extension with an element `StudyExtension` with a type URI “`urn:mpeg:mpeg-g:metadata:extension:ega:Study`”.

To be compliant with the profile, the following extensions shall be present at either the dataset group metadata level (shall be marked as inheritable), or the dataset metadata level:

- One extension with an element `ExperimentExtension` with a type URI “`urn:mpeg:mpeg-g:metadata:extension:ega:Experiment`”.
- One extension with an element `RunExtension` with a type URI “`urn:mpeg:mpeg-g:metadata:extension:ega:Run`”.

Each Sample metadata shall have one extension with an element `SampleExtension` with a type URI “`urn:mpeg:mpeg-g:metadata:extension:ega:Sample`”

Furthermore, to be compliant, the following restrictions have to be observed:

- description element in the Dataset Group metadata schema is mandatory;
- the TaxonID shall be present and equal to 9096 (human).

The contents of `SRA.common.xsd` and `SRA.run.xsd` can be found in Reference [1].

This profile extends the MPEG-G Dataset core metadata to match the Run schema used by EGA. The file element from SRA's run metadata schema is not needed in MPEG-G because the metadata is placed within the element it refers to.

6.7.3 Example of metadata profile — Genomic data commons

The genomic data commons (GDC) classifies the information in projects, which contain the information of multiple cases, represented as a collection of files of different types.

The ISO/IEC 23092 series dataset group shares characteristics with the GDC projects. From all types of information stored in the GDC repository, the ISO/IEC 23092 series only supports raw sequencing analysis. Therefore, the metadata profile designed to allow interoperability between GDC and the ISO/IEC 23092 series only supports that data element from the core metadata.

If the GDC metadata profile is active, the dataset group is considered equivalent to the GDC's project.

This profile is identified with the URI: “`urn:mpeg:mpeg-g:metadata:profile:gdc`”.

When this profile is active, the following extensions are required:

- One extension of type `ProjectsExtensionType` as specified in Annex A.1, identified with a type URI “`urn:mpeg:mpeg-g:metadata:profile:gdc:ProjectsExtensionType`”, listed in the dataset group metadata extensions specified in ISO/IEC 23092-1.
- One extension of type `CaseExtensionType` as specified in Annex A.2, identified with a type URI “`urn:mpeg:mpeg-g:metadata:profile:gdc:CaseExtensionType`”, listed in the dataset metadata extensions specified in ISO/IEC 23092-1. Alternatively, the extension can be provided as an inheritable extension in the dataset group metadata extensions list.
- One extension of type `ClinicalExtensionType` as specified in Annex A.2, identified with a type URI “`urn:mpeg:mpeg-g:metadata:profile:gdc:ClinicalExtensionType`”, listed in the dataset metadata extensions specified in ISO/IEC 23092-1. Alternatively, the extension can be provided as an inheritable extension in the dataset group metadata extensions list.

To be compliant with profile “`urn:mpeg:mpeg-g:metadata:profile:gdc`” all UUID elements shall be present, either in plaintext or encrypted form.

7 Protection metadata

7.1 General

ISO/IEC 23092-1 defines `gen_info` structures supporting the protection of the genomic information at different layers of the hierarchy, namely the elements `dgpr`, `dtpr`, `aupr`, and `dspr`, protecting the dataset group, dataset, descriptor stream and access unit respectively as shown in [Table 4](#). These `gen_info` structures provide information to implement data protection in terms of confidentiality (encryption), integrity verification (digital signature) and privacy rules enforcement. Such information is coded in the form of XML documents as specified in [Table 4](#).

This clause is divided into the three main aspects of protection: encryption, privacy rules and integrity. The protection of specific elements in the metadata is specified in [clause 6.5](#).

Table 4 — Content of the protection boxes specified in ISO/IEC 23092-1

| Content of the <code>gen_info</code> structure specified in ISO/IEC 23092-1 | Key | Value |
|-----------------------------------------------------------------------------|------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| DG_protection_value | dgpr | coded representation of the XML document specified in A.3 . The decoding process to retrieve the XML document from the coded representation is specified in Clause 9 . |
| DT_protection_value | dtpr | coded representation of the XML document specified in A.4 . The decoding process to retrieve the XML document from the coded representation is specified in Clause 9 . |
| AU_protection_value | aupr | coded representation of the XML document specified in A.5 . The decoding process to retrieve the XML document from the coded representation is specified in Clause 9 . |
| DS_protection_value | dspr | coded representation of the XML document specified in A.6 . The decoding process to retrieve the XML document from the coded representation is specified in Clause 9 . |

7.2 Encryption of `gen_info` elements and blocks

7.2.1 General

This subclause specifies how the encryption parameters convey the protection information of some of the `gen_info` elements specified in ISO/IEC 23092-1. A protection `gen_info` structure supporting coded data protection conveys the information on encryption applied to some of its sibling boxes and other `gen_info` structures supporting the protection of a layer immediately below. This information is represented with a list of `EncryptionParameters` elements (of type `EncryptionParametersType`) as specified in [Annex A.3](#).

7.2.2 `EncryptionParameters` carried in dataset group protection

Any number of `EncryptionParameters` elements can be present in the dataset group protection box and shall use one of the following URIs for the `encryptedLocations` attribute, where each URI refers to a specific `gen_info` box as specified below (text within curly brackets, including the curly brackets themselves, shall be replaced by some alphanumeric sequence compliant with the description of each entry).

7.2.2.1 URIs for `gen_info` boxes

7.2.2.1.1 URIs structure for `gen_info` boxes

7.2.2.1.1.1 General

The following URIs shall correspond to specific `gen_info` boxes:

```
refmeta/{id}
metadata
dataset/{id}/protection
```

7.2.2.1.1.2 URI `refmeta/{id}`

This URI shall correspond to all but the two first bytes of the `Value` field (defined in ISO/IEC 23092-1:2019, subclause 6.3) of the `gen_info` structure with key “`rfmd`” (i.e. reference metadata defined in ISO/IEC 23092-1:2019, subclause 6.5.1.4, the two bytes corresponding to values `dataset_group_ID` and `reference_ID`), identified by the field `reference_ID`, and which is stored in the same dataset group as the current dataset group protection. [Table 7](#) describes the type and semantics of the parameter of the URI.

Table 5 — Type and semantics of parameters of the URI `refmeta/{id}`

| Parameter | Type | Semantics |
|-----------------|------------------|-------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>id</code> | unsigned integer | ID of the reference metadata to which the URI correspond. It corresponds to <code>reference_ID</code> defined in ISO/IEC 23092-1:2019, Table 13 |

7.2.2.1.1.3 URI `metadata`

This URI shall correspond to the bytes of the `Value` field (defined in ISO/IEC 23092-1:2019, subclause 6.3) of the `gen_info` box with key “`dgmd`” (i.e. dataset group metadata defined in ISO/IEC 23092-1:2019, subclause 6.5.1.6), stored in the same dataset group as the current dataset group protection.

7.2.2.1.1.4 URI `dataset/{id}/protection`

This URI shall correspond to the bytes of the `Value` field (defined in ISO/IEC 23092-1:2019, subclause 6.5.3.5) of the `gen_info` box with key “`dtpr`” (i.e. dataset protection defined in ISO/IEC 23092-1:2019, subclause 6.5.2.4), stored in the dataset identified by `dataset_ID`, and which belongs to the current dataset group. [Table 10](#) describes the type and semantics of the parameter of the URI.

Table 6 — Type and semantics of parameters of the URI `dataset/{id}/protection`

| Parameter | Type | Semantics |
|-----------------|------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>id</code> | unsigned integer | Identifier of the dataset to which the dataset protection to be retrieved belongs. It corresponds to the field <code>dataset_ID</code> defined in ISO/IEC 23092-1:2019, Table 19. |

7.2.2.1.2 URIs processing

The `EncryptionParameters` elements with either one of the URIs listed above shall adhere to the following rules:

- the `IV` element shall be present;
- the `TAG` element shall be present if the cipher is used in GCM mode (as described in [Table 14](#));
- the `configurationID` attribute shall not be present.

The ciphertext of the `gen_info` boxes identified by the URIs specified in this subclause shall be decrypted using the process specified in [7.2.5.1](#).

7.2.3 EncryptionParameters carried in dataset protection

7.2.3.1 General

Any number of `EncryptionParameters` elements can be present in the dataset protection box and shall use one of the URIs defined in 7.2.3.2 or in 7.2.3.3 for the `encryptedLocations` attribute, where each URI refers either to a specific `gen_info` box or to a collection of access units as specified below. These URIs can contain some placeholders identified by a text enclosed in curly brackets. These placeholders (including curly brackets) shall be replaced by an alphanumeric value as specified below.

7.2.3.2 URIs for `gen_info` boxes

7.2.3.2.1 URIs structure for `gen_info` boxes

7.2.3.2.1.1 General

The following URIs shall correspond to specific `gen_info` boxes:

```
pars/{id}
metadata
descstream/{classId}/{descriptorId}/protection
au/{auType}/{sequenceName}/{accessUnitId}/protection
au/unaligned/{accessUnitId}/protection
```

7.2.3.2.1.2 URI `pars/{id}`

This URI shall correspond to the bytes of the element “`encoding_parameters()`” (defined in ISO/IEC 23092-1:2019, Table 23) in the `gen_info` with key “`pars`” (i.e. dataset parameter set defined in ISO/IEC 23092-1:2019, subclause 6.5.2.5), stored in the same dataset as the current Dataset Protection, identified by `descriptorId`. Table 7 describes the type and semantics of the parameter of the URI.

Table 7 — Type and semantics of parameters of the URI `pars/{id}`

| Parameter | Type | Semantics |
|--------------------------|------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>parameterId</code> | unsigned integer | ID of the <code>encoding_parameters</code> to which the URI correspond. It corresponds to <code>parameter_set_ID</code> defined in ISO/IEC 23092-1:2019, Table 23. |

7.2.3.2.1.3 URI `metadata`

This URI shall correspond to the bytes of the `Value` field (defined in ISO/IEC 23092-1:2019, subclause 6.3) of the `gen_info` box with key “`dtmd`” (i.e. dataset metadata defined in ISO/IEC 23092-1:2019, subclause 6.5.2.3), stored in the same dataset as the current dataset protection.

7.2.3.2.1.4 URI `descstream/{classId}/{descriptorId}/protection`

This URI shall correspond to the bytes of the `Value` field (defined in ISO/IEC 23092-1:2019, subclause 6.3) of the `gen_info` box with key “`dspr`” (i.e. descriptor stream protection defined in ISO/IEC 23092-1:2019, subclause 6.6.4.3), stored in the same dataset as the current Dataset Protection, identified by `classId` and `descriptorId`. Table 8 describes the type and semantics of the parameters of the URI.

Table 8 — Type and semantics of parameters of the URI `descstream/{classId}/{descriptorId}/protection`

| Parameter | Type | Semantics |
|----------------------|------------------|-------------------------------------------------------------------------------------------------------------------------------------------|
| <code>classId</code> | unsigned integer | ID of the class to which the URI correspond. It corresponds to the field <code>class_ID</code> defined in ISO/IEC 23092-1:2019, Table 33. |

Table 8 (continued)

| Parameter | Type | Semantics |
|--------------|------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------|
| descriptorId | unsigned integer | ID of the descriptor to which the URI correspond. It corresponds to the field <code>descriptor_ID</code> defined in ISO/IEC 23092-1:2019, Table 33. |

7.2.3.2.1.5 URI `au/{auType}/{sequenceName}/{accessUnitId}/protection`

This URI shall correspond to the bytes of the the `value` field (defined in ISO/IEC 23092-1:2019, subclause 6.5.3.5) of the `gen_info` box with key “aupr” (i.e. access unit protection) stored in the access unit, stored in the same dataset as the current Dataset Protection, identified by `auType`, `sequenceName` and `accessUnitId`. Table 13 describes the type and semantics of the parameters of the URI.

Table 9 — Type and semantics of parameters of the URI `au/{auType}/{sequenceName}/{accessUnitId}/protection`

| Parameter | Type | Semantics |
|---------------------------|------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>auType</code> | unsigned integer | Identifier of the class to which the URI correspond. It corresponds to the field <code>class_ID</code> defined in ISO/IEC 23092-1:2019, Table 25. The value shall be greater or equal than 1 and strictly less than 6. |
| <code>sequenceName</code> | st(v) | Textual identifier of the reference sequence the access unit to which the access unit protection to be retrieved is mapped to. It corresponds to <code>sequence_name</code> specified in ISO/IEC 23092-1:2019, Table 10. |
| <code>accessUnitId</code> | unsigned integer | Identifier of the access unit to which the access unit protection to be retrieved belongs. It corresponds to the field <code>access_unit_ID</code> defined in ISO/IEC 23092-1:2019, Table 25. |

7.2.3.2.1.6 URI `au/unaligned/{accessUnitId}/protection`

This URI shall correspond to the bytes of the the `value` field (defined in ISO/IEC 23092-1:2019, subclause 6.5.3.5) of the `gen_info` box with key “aupr” (i.e. access unit protection) stored in the access unit, stored in the same dataset as the current Dataset Protection, for which the associated `AU_type` defined in ISO/IEC 23092-1:2019, Table 25 is equal to the parameter `accessUnitId` provided. Table 10 describes the type and semantics of the parameter of the URI.

Table 10 — Type and semantics of parameters of the URI `au/unaligned/{accessUnitId}/protection`

| Parameter | Type | Semantics |
|---------------------------|------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>accessUnitId</code> | unsigned integer | Identifier of the access unit to which the access unit protection to be retrieved belongs. It corresponds to the field <code>access_unit_ID</code> defined in ISO/IEC 23092-1:2019, Table 25. |

7.2.3.2.2 URIs processing

The `EncryptionParameters` elements with either one of the URIs listed above shall adhere to the following rules:

- the `IV` element shall be present;
- the `TAG` element shall be present if the cipher is used in GCM mode (as described in Table 14);
- the `configurationID` attribute shall not be present.

The ciphertext of the `gen_info` boxes identified by the URIs specified in this subclause shall be decrypted using the process specified in 7.2.5.1.

7.2.3.3 URIs for collection of access units

7.2.3.3.1 URIs structure for access units

7.2.3.3.1.1 General

The following URIs shall correspond to collections of access units:

```
genomic_region/{auType}/{sequenceName}/{start}/{end}
unaligned/{signature}
unaligned/{signature}
```

7.2.3.3.1.2 URI genomic_region/{auType}/{sequenceName}/{start}/{end}

This URI shall correspond to access units containing at least one read mapping between `start` and `end` (included) on the reference sequence identified by `sequenceName`. [Table 11](#) describes the type and semantics of the parameters of the URI.

Table 11 — Type and semantics of parameters of the URI `genomic_region/{auType}/{sequenceName}/{start}/{end}`

| Parameter | Type | Semantics |
|---------------------------|----------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>auType</code> | unsigned integer equal or greater than 0 and less than 6 | Type of access units to be retrieved. When set to 0, all access units having <code>AU_type</code> (as specified in ISO/IEC 23092-1:2019, Table 25) less than 6 shall be retrieved. |
| <code>sequenceName</code> | <code>st(v)</code> | Textual identifier of the reference sequence the access units to be retrieved are mapped to. It corresponds to <code>sequence_name</code> specified in ISO/IEC 23092-1:2019, Table 10. |
| <code>start</code> | unsigned integer | Left-most position of the interval to be considered for access units retrieval. It shall be less than or equal to <code>end</code> . |
| <code>end</code> | unsigned integer | Right-most position of the interval to be considered for access units retrieval. It shall be greater than or equal to <code>start</code> . |

7.2.3.3.1.3 URI `unaligned/{signature}`

This URI shall correspond to access units having `AU_type` (as specified in ISO/IEC 23092-1:2019, Table 25) equal to 6 and at least one of the `U_cluster_signature` fields (specified both in ISO/IEC 23092-1:2019, Table 25 and in Table 31) matching the value of `signature`. [Table 12](#) describes the type and semantics of the parameter of the URI.

Table 12 — Type and semantics of parameters of the URI `unaligned/{signature}`

| Parameter | Type | Semantics |
|------------------------|--------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>signature</code> | <code>st(v)</code> | At least one of the <code>U_cluster_signature</code> fields (specified both in ISO/IEC 23092-1:2019, Table 25 and in Table 31) of the retrieved access units shall match this value. |

7.2.3.3.1.4 URI `label_selection/{labelId}`

This URI shall correspond to access units containing at least one read mapping in at least one of the genomic regions listed in the label identified by `labelId` as specified in ISO/IEC 23092-1:2019, subclause 6.5.1.5.4. [Table 13](#) describes the type and semantics of the parameter of the URI.

Table 13 — Type and semantics of parameters of the URI `label_selection/{labelId}`

| Parameter | Type | Semantics |
|----------------------|------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>labelId</code> | unsigned integer | Identifier of the label to be considered for retrieval. It corresponds to <code>label_ID</code> specified in ISO/IEC 23092-1:2019, subclause 6.5.1.5.4. |

7.2.3.3.2 URIs processing

For all access units belonging to the collection resolved by any of the previous URIs, the bytes that shall be decrypted are:

- If the access unit contains a `gen_info` with key “auin”, the bytes corresponding to the `Value` field (defined in ISO/IEC 23092-1:2019, subclause 6.3) of the `gen_info` with key “auin” (the access unit information defined in ISO/IEC 23092-1:2019, subclause 6.5.3.4)
- In AUC mode:
 - the bytes of the `Value` field (defined in ISO/IEC 23092-1:2019, subclause 6.3) of the access unit, discarding all bytes prior to and including the last byte of the access unit protection

In DSC mode:

- all bytes corresponding to the associated `blocks` structures (as specified in ISO/IEC 23092-1), concatenated, in growing order of `descriptor_ID` defined in ISO/IEC 23092-1:2019, Table 19.

Both decryptions shall be performed independently, using their own IVs and TAGs (present only with GCM mode) signaled in the `AccessUnitEncryptionParameters` element present in the associated AU protection box (specified in [A.5](#)).

The `EncryptionParameters` elements with either of the URIs listed above shall adhere to the following rules:

- the `IV` element shall not be present;
- the `TAG` element shall not be present;
- the `configurationID` attribute shall be present. If an access unit belongs to one of the collection resolved by the URIs listed above, the `AccessUnitEncryptionParameters` element of the access unit protection shall contain one or more `wrappedKey` elements each referring to the `configurationID` of one of the `EncryptionParameters` element resolving to the access unit. The key associated to each of these `EncryptionParameters` shall allow to unwrap the corresponding `wrappedKey`.

7.2.4 Key retrieval

7.2.4.1 General

Across all `DatasetGroupProtection`, `DatasetProtection`, `AccessUnitProtection` or `DescriptorStreamProtection` elements present in one file compliant to the ISO/IEC 23092 series, multiple `KeyTransportation` elements defining a key with the same name may be present, and all of them shall encapsulate the same value, albeit with different parameters. The name of the key is encapsulated in the `key_name` element of the `KeyTransportation` element. A `KeyTransportation` element can transport the key in three different ways, as described in the three following subclauses. Additionally, a key can be referenced by a name not appearing in the `key_name` element of any `KeyTransportation` element; in that case, the key shall be transported through another channel. Some examples are provided in [Annex D](#).

7.2.4.2 Key derivation

The information is provided in a `KeyDerivation` element as defined in [Annex A.3](#). The value of the key shall be obtained by executing the PBKDF2 algorithm as defined in RFC 8018. The elements defined in the `KeyDerivation` tag indicate the parameters to be used as input for the PBKDF2 algorithm. The value of the `password` used in the execution of the PBKDF2 algorithm shall be retrieved as the value of the key having the name equal to the value of the `passwordName` element, and be retrieved as specified in subclause [7.2.4](#).

7.2.4.3 Symmetric encrypted key

The information is provided in a `KeySymmetricWrap` element as defined in Annex A.3. The key in wrapped form is stored in the `wrappedKey` element of the `KeySymmetricWrap`. The value of key to be used in the unwrapping procedure is the value of the key having the name equal to the value of the `kek` element, and is to be retrieved as if it was a key, as explained in subclause 7.2.4. The key is to be unwrapped according to the RFC 3394 algorithm, the method to be used is determined by the length of the key referenced by the value of the `kek` element, and the expected key length (derived from the selected cipher as specified in Table 14).

7.2.4.4 Asymmetric encrypted key

The information is provided in a `KeyAsymmetricWrap` element as defined in Annex A.3. The key shall be obtained by applying the decrypting process defined in RFC 8017 for `RSAPKCS1v1_5`. The ciphertext to be decrypted is the content of the element `wrappedKey`, the key to be used is the private key belonging to the asymmetric key pair identified by the value of the element `publicKeyName`, and the hash method to be used is indicated by the `hashFunction` element. The `Label` parameter used in the decryption process of RFC 8017 shall be provided as an empty string. The MGF function to be used shall be the MGF1 as defined in RFC 8017, and the hash function to be used in the MGF function is identified by the `maskGenerationHashFunction` element.

7.2.5 Decryption

7.2.5.1 Decryption of `gen_info` elements

This clause specifies the decryption process when the value of the `encryptedLocations` element of the XML `EncryptionParameters` element matches one of the URI syntaxes defined in subclauses 7.2.2 and 7.2.3.2.

In order to decrypt the associated content, the key with name equal to the value of the `key` element in the `EncryptionParameter` shall be retrieved as specified in 7.2.4. The cipher is identified by the `cipher` element value of the `EncryptionParameter`. The key shall have exactly the size required by the cipher.

Table 14 — List of supported ciphers and key sizes

| Cipher | Size required |
|---------------------------------------|---------------|
| urn:mpeg:mpeg-g:protection:aes128-ctr | 128 bits |
| urn:mpeg:mpeg-g:protection:aes192-ctr | 192 bits |
| urn:mpeg:mpeg-g:protection:aes256-ctr | 256 bits |
| urn:mpeg:mpeg-g:protection:aes128-gcm | 128 bits |
| urn:mpeg:mpeg-g:protection:aes192-gcm | 192 bits |
| urn:mpeg:mpeg-g:protection:aes256-gcm | 256 bits |

The cipher used is AES as defined in FIPS 197, the length of the key used is either 128, 192 or 256 as indicated in the URI. The “CTR” in the cipher name refers to the Counter mode of operations as defined in 800-38A. The “GCM” refers to the Galois Counter Mode mode of operations as defined in 800-38D. If the GCM mode is used, then the `EncryptionParameters` shall have an element `TAG`.

Upon decryption, the ciphertext bytes shall be replaced in-place with the decrypted bytes.

7.2.5.2 Decryption of blocks and access unit information

This clause specifies the decryption process when the value of the `encryptedLocations` element of the XML `EncryptionParameters` element matches one of URI defined in 7.2.3.3.

The ciphers used are the same as in 7.2.5.1.

When an access unit belongs to a collection which an `EncryptionParameters` resolved to, then it is to be decrypted as below:

- (1) Retrieve from the `EncryptionParameters` element:
 - a. the `key`;
 - b. the `configurationID`.
- (2) Retrieve from the `AccessUnitEncryptionParameters` element present in the associated AU protection box (specified in A.5):
 - a. the cipher (possible values are listed Table 14);
 - b. the `wrappedKey` matching the `configurationID` (retrieved in step 1);
 - c. `auinIV`, if the access unit contains an access unit information box;
 - d. `auinTAG`, if the access unit contains an access unit information box and the cipher uses GCM mode;
 - e. `aublockIV`;
 - f. `aublockTAG` if the cipher uses GCM mode.
- (3) Retrieve the decryption key for the current access unit by unwrapping the contents of `wrappedKey` by applying method specified in RFC 3394 using the `key` (the actual method to use is derived from the length of the unwrapping key identified by the `EncryptionParameters`' `key` and the expected key length derived from the selected cipher as specified in Table 14).
- (4) If the access unit contains an access unit information box, decrypt using the cipher and the `key` (obtained in the previous step), `auinIV` and `auinTAG` (if present), the `Value` field of the `auin_gen_info` box of the access unit and replace the ciphertext with the plaintext in place.
- (5) Decrypt using the cipher and the `key` (obtained in the previous step), `aublockIV` and `aublock_TAG` (if present),
 - a. if in AUC mode:

Decrypt using the cipher and the `key` (obtained in the previous step), `aublockIV` and `aublockTAG` (if present), all bytes after the last byte of the access unit protection (defined in ISO/IEC 23092-1:2019, Table 24), and replace the ciphertext bytes with the plaintext bytes in place.
 - b. Else if in DSC mode:

Decrypt using the cipher and the `key` (obtained in the previous step), `aublock_IV` and `aublock_TAG` (if present), all bytes corresponding to the associated `blocks` payloads (resolved as specified in ISO/IEC 23092-1), concatenated, in growing order of `descriptor_ID` defined in ISO/IEC 23092-1:2019, Table 19, and replace the ciphertext bytes with the plaintext bytes in place, replacing the content of one `block` associated to the access unit (as specified in ISO/IEC 23092-1) at a time, without modifying the byte ordering (the number of bytes to replace for a specific block is the one derived from the Master Index Table as specified in ISO/IEC 23092-1).

7.3 Privacy rules for the use of the genomic information

7.3.1 General

The `privacy_rules` tag is defined in the schema for protection metadata at the dataset group and dataset levels of the hierarchy. The `privacy_rules` tag shall be a valid policy element specified according to the OASIS, eXtensible Access Control Markup Language (XACML) Version 3.0 specification. Exporting this tag as the root element of a new document, a valid policy document is obtained.

The `privacy_rules` specify who can execute a given action and under which conditions. The information is conveyed according to the XACML specification.

In the privacy policies, the XACML `actions` are referred to by using the names defined in the different subclauses of subclause 8.1. Before executing an API action, the generated request shall be checked against the datasetgroup policy and possibly the dataset policy. Each action indicates, in its Authorization paragraph, whether the dataset policy should be checked. If there is no Authorization paragraph, the dataset policy shall not be checked. If it is the case, before submitting the request to the dataset policy, one new attribute is added to the list of attributes of the request: the new attribute shall have an `AttributeId` equal to `urn:mpeg:mpeg-g:protection:granted_by_dataset_group`, a `DataType` equal to <https://www.w3.org/2001/XMLSchema#boolean>, and a value of `true` if the request yields a Permit result at the datasetgroup level, otherwise `false`. If no `privacy_rules` element is provided in the protection box, then all actions are granted.

Actions are referred in privacy policies by using the name defined in the different subclauses of subclause 8.1. Each XACML action is described as an XACML `attributeValue` with category `"urn:oasis:names:tc:xacml:3.0:attribute-category:action"` and `attributeId` `"urn:oasis:names:tc:xacml:1.0:action:action-id"`.

The rule applies to a `resource` (as defined in XACML) determined from the definition of the function in subclause 8.1, and where the rule is stored (i.e. Dataset Group, Dataset).

Rules apply to `subject` or `role`.

`subject` is described as an XACML `attributeValue` with category `"urn:oasis:names:tc:xacml:1.0:subject-category:access-subject"` and `attributeId` `"urn:oasis:names:tc:xacml:1.0:subject:subject-id"`.

`role` is described as an XACML `attributeValue` with category `"urn:oasis:names:tc:xacml:1.0:subject-category:access-subject"` and `attributeId` `"urn:oasis:names:tc:xacml:3.0:example:attribute:role"`.

If the entity providing the file requires the use of further attributes, the corresponding names, types and semantics shall be provided through another channel.

The API methods, described in subclause 8.1, have a set of possible parameters, each with a specific range of valid values. The privacy rules can further limit the range of accepted input parameters using conditions. The conditions shall refer to the action input parameters by referring to attributes whose `AttributeId` is equal to the name of the parameter and whose type corresponds to the type of the parameter. The request generated to test the authorization to perform a function shall include one attribute per input parameter with the `AttributeId` and type, as specified before.

In the case of `getData` interfaces (subclauses 10.3.6.3, 10.3.6.4, 10.3.6.5, 10.3.6.6), the privacy rules concerning access units are derived as follows:

- In the case of an access unit of unaligned data, the `getDataBySignature` privacy rule is tested for each of the access unit signatures. If the action is permitted for each signature, then the `getDataBySignature` is permitted for that access unit, otherwise it is denied.
- In the case of an access unit of aligned data, the `getDataByFilter` privacy rule is tested, populating the filter with the type, start and end position of the access unit.

Annex B.5 contains examples of privacy rules and authorization requests. Subclause 7.3.2 presents an example of a use case where privacy rules and authorization requests presented in Annex B.5 are used.

7.3.2 Example of use of privacy rules

This subclause provides an example use case to illustrate when and how privacy rules can be used to protect genomic information.

Researchers have already identified some regions of the genome indicating that there is a predisposition to suffer Alzheimer's disease. When trying to access that information, one should consider where these

regions are, taking into account their positions inside human genome. Regions identifying Alzheimer's disease predisposition may be protected from unauthorized access by using encryption techniques combined with privacy rules. Therefore, only users authorized by the rules can perform operations over protected regions.

In order to give permission for accessing these protected regions (or some of them), an organization may define privacy policies indicating which person or role can view the genomic regions and the conditions that have to be accomplished.

The same policy may contain different rules to give different permissions to different subjects. When access to one of these regions is requested, the rule or rules applying are evaluated, accepting or denying the corresponding permission.

As an example, a policy may contain several rules. The first rule in the policy may provide permission to perform any operation to researchers. The second one may provide permission to view part of a dataset to practitioners under an emergency situation. To do so, the API operation `GetDataBySimpleFilter` is defined as the action permitted by the rule, and several conditions related to Alzheimer's disease and the quality of the information stored inside the file are involved. With this purpose, the user defining the rule may specify a region number including start and end positions, class of the read, thus identifying. A final rule denies any access to the genomic file. It is the default rule, which is evaluated in case none of the above applies.

7.4 Digital signature of `gen_info` elements and blocks

7.4.1 General

At each level, the protection box may include integrity information in the form of one or more digital signatures. Each signature is provided as an XML Signature element. Detached, Enveloped and Enveloping signatures are supported. If decryption is required, signature verification shall be performed before decryption.

7.4.2 Signatures carried in dataset group protection

Any number of XML Signature elements can be present in the dataset group protection box and shall use one of the following URIs for the `URI` attribute of the `Reference` element when referring to a `gen_info` box. These URIs can contain some placeholders identified by a text enclosed in curly brackets. These placeholders (including curly brackets) shall be replaced by an alphanumeric value as specified below.

- “`refmeta/{id}`” resolves to the bytes of the `Value` field (defined in ISO/IEC 23092-1:2019, subclause 6.3) of the `gen_info` with key “`rfmd`” (i.e. reference metadata), for which the field `reference_ID` defined in ISO/IEC 23092-1:2019, Table 13 is equal to the value which is replacing “`{id}`”, and which is stored in the same dataset group as the current Dataset Group Protection.
- “`metadata`” resolves to the bytes of the `Value` field (defined in ISO/IEC 23092-1:2019, subclause 6.3) of the `gen_info` with key “`dgmmd`” (i.e. dataset group metadata), stored in the same dataset group as the current Dataset Group Protection.
- “`dataset/{id}/protection`” resolves to bytes of the `Value` field (defined in ISO/IEC 23092-1:2019, subclause 6.3) of the `gen_info` with key “`dtpr`” (i.e. dataset protection), stored in the dataset for which the field `dataset_ID` defined in ISO/IEC 23092-1:2019, Table 19 is equal to the value which is replacing “`{id}`”, and which belongs to the current Dataset Group.

7.4.3 Signatures carried in dataset protection

7.4.3.1 General

Any number of XML Signature elements can be present in the dataset protection box and shall use one of the following URIs for the `URI` attribute of the `Reference` element when referring to a `gen_info` box or the content of an access unit. These URIs can contain some placeholders identified by the text enclosed

in curly brackets. These placeholders shall be replaced by an alphanumeric value as specified below. The bytes are signed as transmitted, regardless of encryption.

7.4.3.2 URIs for `gen_info` boxes

- “`pars/{id}`” resolves to the bytes of the `Value` field (defined in ISO/IEC 23092-1:2019, subclause 6.3) of the `gen_info` with key “`pars`” (i.e. dataset parameter set), for which the field `parameter_set_ID` defined in ISO/IEC 23092-1:2019, Table 23 is equal to the value which is replacing “`{id}`”, and which is stored in the same dataset as the current Dataset Protection.
- “`metadata`” resolves to bytes of the `Value` field (defined in ISO/IEC 23092-1:2019, subclause 6.3) of the `gen_info` with key “`dtmd`” (i.e. dataset metadata), stored in the same dataset as the current Dataset Protection.
- “`descstream/{classId}/{descriptorId}/protection`” resolves to the `Value` field (defined in ISO/IEC 23092-1:2019, subclause 6.3) of the `gen_info` with key “`dspr`” (i.e. stream protection) stored in the stream for which the field `class_ID` defined in ISO/IEC 23092-1:2019, Table 19 is equal to the value which is replacing “`{classId}`”, the field `descriptor_ID` defined in ISO/IEC 23092-1:2019, Table 19 is equal to the value which is replacing “`{descriptorId}`”, and which belongs to the current Dataset.
- “`au/{auType}/{sequenceName}/{accessUnitId}/protection`” resolves to the `Value` field (defined in ISO/IEC 23092-1:2019, subclause 6.3) of the `gen_info` with key “`aupr`” (i.e. access unit protection) stored in the access unit for which the associated `AU_type` defined in ISO/IEC 23092-1:2019, Table 25 is equal to the value which is replacing “`{auType}`”, the associated `sequence_name` defined in ISO/IEC 23092-1:2019, Table 10 is equal to the value replacing “`{sequenceName}`” and the field `access_unit_ID` defined in ISO/IEC 23092-1:2019, Table 25 is equal to the value replacing “`{accessUnitId}`”. The value of `auType` shall be equal or greater than 1 and strictly less than 6.
- “`au/unaligned/{accessUnitId}/protection`” resolves to the `Value` field (defined in ISO/IEC 23092-1:2019, subclause 6.3) of the `gen_info` with key “`aupr`” (i.e. access unit protection) stored in the access unit for which the associated `auType` defined in either ISO/IEC 23092-1:2019, Table 25 or Table 19 (under the name `clid`) is equal to 6, and the field `access_unit_ID` defined in ISO/IEC 23092-1:2019, Table 25 is equal to the value replacing “`{accessUnitId}`”.

7.4.3.3 URIs for collection of access units

Any number of signatutes stored in the dataset protection shall also use any of the following URIs to refer to the content corresponding to a collection of access units

- “`genomic_region/{auType}/{sequenceName}/{start}/{end}`” resolves to one collection of access units to sign. The values replacing “`{start}`” and “`{end}`” shall be numbers such that the first one is strictly smaller than the second one. For all access units such that
 - the `AU_type` defined in either ISO/IEC 23092-1:2019, Table 25 is equal to the value which is replacing “`{auType}`”, or the value replacing “`{auType}`” is equal to “0” and the `AU_type` defined in ISO/IEC 23092-1:2019, Table 25 is strictly smaller than 6.
 - the associated `sequence_name` defined in ISO/IEC 23092-1:2019, Table 10 is equal to the value replacing “`{sequenceName}`”,
 - and the access unit range intersects with the range defined by “`{start}`” (included) and “`{end}`” (included).

The content, resolved by this URI, to sign is obtained by concatenating the following sequences of bytes: for all the access units belonging to the collection resolved by this URI and ordered in growing order of `AU_type`, and `access_unit_ID`

- the bytes corresponding to the `Value` field (defined in ISO/IEC 23092-1:2019, subclause 6.3) of the `gen_info` with key “`auin`” which belongs to the access unit.

- In AUC mode:
 - the bytes of the `Value` field (defined in ISO/IEC 23092-1:2019, subclause 6.3) of the access unit, discarding all bytes prior to, and including the last byte of the access unit protection.
- In DSC mode:
 - all bytes corresponding to the associated `block payloads` (resolved as specified in ISO/IEC 23092-1), organized in increasing order of the `descriptor_ID` (as specified in ISO/IEC 23092-1:2019, Table 33).
- “unaligned/{signature}” resolves to one collection of access units to sign. For all access units such that
 - the `AU_type` defined in ISO/IEC 23092-1:2019, Table 25 is equal to 6
 - At least one of the `U_cluster_signature` field (defined in ISO/IEC 23092-1:2019, Table 25 or in Table 31) is exactly equal to the value replacing “{signature}” and order by access unit id, the following bytes are concatenated (in the order of appearance) to the byte array to sign.

The content resolved by this URI is obtained by concatenating the following bytes: for all the access units belonging to the collection resolved by this URI ordered in growing order of `access_unit_ID`

 - the bytes corresponding to the `Value` field defined in (defined in ISO/IEC 23092-1:2019, subclause 6.3) of the `gen_info` with key “auin” which belongs to the access unit
 - In AUC mode:
 - the bytes of the `Value` field (defined in ISO/IEC 23092-1:2019, subclause 6.3) of the access unit, discarding all bytes prior to the last byte of the access unit protection.
 - In DSC mode:
 - all bytes corresponding to the associated `block payloads` (resolved as specified in ISO/IEC 23092-1), in growing order of `descriptor_ID` defined in ISO/IEC 23092-1:2019, Table 19.

7.4.4 Signatures carried in access unit protection

Any number of signatures stored in the access unit protection shall also use one of the following URIs to refer to a `gen_info` structure or encoded genomic information:

- “auhd” resolves to the `Value` field (defined in ISO/IEC 23092-1:2019, subclause 6.3) of the `gen_info` with key “auhd” (i.e. access unit header) stored in the same access unit container as the access unit protection.
- “blocks” resolved to the bytes of the `Value` field (defined in ISO/IEC 23092-1:2019, subclause 6.3) of the access unit, discarding all bytes prior to the last byte of the access unit protection.
- “auin” resolves to the the bytes corresponding to the `Value` field defined in (defined in ISO/IEC 23092-1:2019, subclause 6.3) of the `gen_info` with key “auin” which belongs to the access unit.

7.4.5 Signatures carried in descriptor stream protection

Any number of signatures stored in the descriptor stream protection shall also use one of the following URIs to refer to a `gen_info` structure or encoded genomic information:

- “dshd” resolves to the `Value` field (defined in ISO/IEC 23092-1:2019, subclause 6.3) of the `gen_info` with key “dshd” (i.e. descriptor stream header) stored in the same descriptor stream container as the descriptor stream protection.

- “blocks” resolves to the bytes of the the StreamContainer, discarding all bytes prior to the last byte of the descriptor stream protection.

8 Access unit information

8.1 General

Coded auxiliary fields associated to genomic records are carried by the `AU_information_value` array contained in the `gen_info` element `AU_information` defined in ISO/IEC 23092-1. The decoding process described in [Clause 9](#) applied on `AU_information_value` produces an array of `genAuxRecord` structures specified in subclause 8.2. Each `genAuxRecord` structure contains `numberOfGenAux` sequences of `gen_aux` structures defined in subclause 8.3.

The i^{th} decoded `genAuxRecord` structure shall be associated to the i^{th} decoded genomic record. If a coded genomic record is not associated to any auxiliary field but it is coded in an AU containing at least one genomic record with associated auxiliary fields, then said genomic record is associated with one `genAuxRecord` structure with `numberOfGenAux` set equal to 0. Moreover, if all genomic records coded in an AU have no associated auxiliary fields, the `gen_info` element `AU_information` shall not be present.

8.2 genAuxRecord

The syntax of `genAuxRecord` is defined as follows:

```
struct genAuxRecord
{
    u(8)      numberOfGenAux;

    genAux    auxSet[numberOfGenAux];
}
```

[Table 15](#) shows how a `genAuxRecord` structure is associated to each decoded genomic record (specified in ISO/IEC 23092-2:2019, subclause 13.2) and how auxiliary fields are decoded to each alignment of mapped record segments or unmapped record segments. [Table 15](#) shows as well how to compute `numberOfGenAux`.

Table 15 — Association of auxiliary fields to mapped and unmapped record segments

```
numberOfGenAux= 0
noa = 0
for (; noa < number_of_alignments; noa++) {
    if((noa == 0 ||
        (mapping_pos[noa] != mapping_pos[noa - 1] &&
         ecigar_cmp(ecigar_string[noa][0], ecigar_string[noa-1][0] != 0))){
        aux_fields[noa][0] = aux_Set[numberOfGenAux++]
    }
    else{
        aux_fields[noa][0] = aux_fields[noa-1][0]
    }
    if(class_ID != Class_HM){
        for (tSeg=1; tSeg <= number_of_template_segments; tSeg++)

            if(split_alignment[noa][tSeg] == 0) {

                if((noa == 0 ||
                    (mapping_pos[noa] + delta[noa][tSeg] != mapping_pos[noa - 1] +
                     delta[noa-1][tSeg] &&
```


Table 15 (continued)

```

        ecigar_cmp(ecigar_string[noa][tSeg], ecigar_string[noa-1][tSeg] != 0)){
            aux_fields[noa][tSeg] = aux_Set[numberOfGenAux++]
        }
        else{
            aux_fields[noa][tSeg] = aux_fields[noa-1][tSeg]
        }
    }
}
for(    uSeg = numberOfAlignedRecordSegments;
        uSeg < number_of_record_segments; uSeg++){
    aux_fields[0][uSeg] = aux_Set[numberOfGenAux++]
}

```

`ecigar_cmp` compares two e-cigar strings and returns 1 if either the length or the strings are different. Otherwise it returns 0.

`aux_fields[i][j]` contains the auxiliary fields associated to the i^{th} alignments of the j^{th} segment for mapped record segments. In case of unmapped reads, `aux_fields[0][j]` contains the auxiliary fields associated to the j^{th} unmapped record segment.

`number_of_alignments`, `number_of_template_segments` and `number_of_record_segments` are specified in ISO/IEC 23092-2:2019, subclause 13.2.

`numberOfAlignedRecordSegments` is specified in ISO/IEC 23092-2:2019, subclause 10.4.9.

8.3 genAux

The syntax of `genAux` is defined as follows:

```

struct genAux
{
    u(8)    numberOfTags;
    genTag  auxFields[numberOfTags];
}

```

`numberOfTags` represents the number of auxiliary fields associated with the current genomic record.

`auxFields` is a list of `genTag` structures containing the values of the auxiliary fields.

8.4 genTag

The syntax of `genTag` is defined as follows:

```

struct genTag
{
    c(2)  key;
    u(4)  type;
    u(16) length;
    type  value[];
    if(!byte_aligned()) f(4) nestingZeroBit;
}

```

Key is a two-characters identifier of the tag, Type an identifier of the type of data contained in Value, Length is the number of elements contained in the value[] array, and value is an array that contains the auxiliary field value. The bit order of each element in the Value array is specified to start with the most significant bit (MSB) and proceed to the least significant bit (LSB).

The padding with 4 bits (nestingZeroBit) is necessary when Type is not equal to 7 or when it is equal to 7 and length is an even number.

The field type shall take one of the values specified in Table 16.

Table 16 — Values for the Type field of the gen_tag structure

| Type | data type | type | Description |
|------|----------------------------------------------------------------------|-------|----------------------------------------------------------------------------------------------------------------------|
| 0 | Signed 32 bit integer | i(32) | It corresponds to type "i" in the SAM specification. |
| 1 | string of ASCII characters | c(1) | It corresponds to type "Z" in the SAM specification. If Length is equal to 1 this corresponds to the SAM type "A" |
| 2 | Unsigned 8 bit integer | u(8) | It corresponds to type "C" in the SAM specification. |
| 3 | Signed 8 bit integer | i(8) | It corresponds to type "c" in the SAM specification. |
| 4 | Unsigned 16 bit integer | u(16) | It corresponds to type "S" in the SAM specification. |
| 5 | Signed 16 bit integer | i(16) | It corresponds to type "s" in the SAM specification. |
| 6 | Unsigned 32 bit integer | u(32) | It corresponds to type "I" in the SAM specification. |
| 7 | 4-bit hexadecimal code. | u(4) | It corresponds to type "H" in the SAM specification. |
| 8 | 32 bit single precision floating-point as specified by IEEE 754-2008 | f32 | It corresponds to type "f" in the SAM specification. |
| 9 | 64 bit double precision floating-point as specified by IEEE 754-2008 | f64 | It corresponds to type "d" in the SAM specification. |

9 Decoding process for metadata

9.1 General

This clause describes the decoding process of coded information metadata, protection metadata and auxiliary fields¹⁾.

Input and output of the process are specified in Table 17.

1) The decoding process relies on the implementation of the LZMA SDK version 18.05 published at <https://sourceforge.net/projects/sevenzip/files/LZMA%20SDK/lzma1805.7z/download> on 2018-05-01. Any other process producing the same output from the same input can be equally considered compliant to this document. At the same time, complying implementations are not expected to follow the exact algorithm used by this reference procedure.

Table 17 — Input and output of the decoding process specified in this clause

| Input | Output | Comments |
|--------------------------------------------|-----------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------|
| Information metadata | | |
| DG_metadata_value | Dataset group metadata XML document specified in subclause 6.2. | DG_metadata_value is contained in DG_metadata as specified in ISO/IEC 23092-1 |
| reference_metadata_value | Reference metadata XML document specified in subclause 6.3. | reference_metadata_value is contained in reference_metadata as specified in ISO/IEC 23092-1 |
| DT_metadata_value | Dataset metadata XML document specified in subclause 6.4. | DT_metadata_value is contained in DT_metadata as specified in ISO/IEC 23092-1 |
| Protection metadata | | |
| DG_protection_value | protection metadata XML document specified in Clause 7. | DG_protection_value is contained in DG_protection as specified in ISO/IEC 23092-1 |
| DT_protection_value | protection metadata XML document specified in Clause 7. | DT_protection_value is contained in DT_protection as specified in ISO/IEC 23092-1 |
| DS_protection_value | protection metadata XML document specified in Clause 7. | DS_protection_value is contained in DS_protection as specified in ISO/IEC 23092-1 |
| AU_protection_value | protection metadata XML document specified in Clause 7. | AU_protection_value is contained in AU_protection as specified in ISO/IEC 23092-1 |
| Access unit information | | |
| AU_information_value specified in Clause 8 | genAuxRecord [num_genomic_records] | genAuxRecord is specified in subclause 8.2 num_genomic_records represents the number of genomic records carried by the decoded access unit |

In the description of this decoding process, the input arrays listed in Table 17 are referred to as `inputBuffer` of size `inputSize` and the output arrays as `outputBuffer` of size `outputSize`.

Coded metadata are represented as a LZMA compliant byte array with a 13 bytes header and a payload. The header is specified in Table 18.

Table 18 — Header of LZMA coded metadata

| Header | |
|--------------|-------|
| params | u(8) |
| dictSize | u(32) |
| unpackedSize | u(64) |

The **params** byte is used to calculate the `lc`, `pb` and `lp` members of the `lzmaProps` structure specified in subclause 9.2.

dictSize and **unpackedSize** are stored in the `lzmaProps` structure specified in subclause 9.2.

9.2 Initialization of parameters

9.2.1 General

This subclause specifies the constants and data structures containing the parameters governing the decoding process described in subclause 9.3.

9.2.2 Properties

LZMA properties are contained in the following structure.

```
struct lzmaProps{
    u(32) dictSize
    u(8)  lc
    u(8)  pb
    u(8)  lp
    u(64) uncompressedSize
}
```

lc is the number of high bits of the previous byte to use as a context for literal encoding (the default value used by the LZMA SDK is 3).

lp is the number of low bits of the dictionary position to include in literal_pos_state (the default value used by the LZMA SDK is 0).

pb is the number of low bits of the dictionary position to include in pos_state (the default value used by the LZMA SDK is 2).

9.2.3 Parameters

LZMA parameters are contained in the following structure.

```
struct lzmaParams{
    lzmaPropsp
    u(16) numProbs
    u(16) probs[numProbs]
    u(16) probs_1664[numProbs]
    u(8)  dic[]
    u(64) dicBufSize
    u(64) dicPos
    u(32) range
    u(32) code
    u(32) processedPos
    u(32) checkDicSize
    u(32) state
    u(32) reps[4]
    u(32) remainLen
    u(8)  buf[]
}
```

prop is a lzmaProps structure specified in this subclause.

numProbs represents the number of probabilities used.

probs is a list of numProbs probabilities.

probs_1664 is a list of numProbs probabilities.

dicBufSize represent the size of the dictionary used by LZMA.

dic[] contains the dictionary used by the LZMA algorithm. The size is represented by dicBufSize.

dicPos is a pointer used to access the dic[] list.

range and **code** are state variable used in the ragencoder engine of the LZMA algorithm.

processedPos represents the last coded match offset away from current pointer.

checkDicSize is a state variable used to manage the size of the dictionary.

state is an integer indicating what the last performed coding operations have been.

reps[4] is a list of variables used to manage the access to the dictionary.

remainLen represents the number of bytes remaining to be decompressed.

buf[] contains the decoded bytes.

9.2.4 Constants

LZMA constants are listed in [Table 19](#).

Table 19 — Constants defined in the decoding process

| Constant Name | Value |
|-----------------------|------------------------------------------|
| kLzmaLitSize | 0x300 |
| kNumBaseProbs | 1984 |
| kStartOffset | 1664 |
| kMatchSpecLenStart | 274 |
| kLenNumLowBits | 3 |
| kLenNumHighBits | 8 |
| kNumStates | 12 |
| kNumStates2 | 16 |
| kNumLitStates | 7 |
| kStartPosModelIndex | 4 |
| kEndPosModelIndex | 14 |
| kNumFullDistances | 2 ⁷ |
| kNumPosSlotBits | 6 |
| kNumLenToPosStates | 4 |
| kMatchMinLen | 2 |
| kNumAlignBits | 4 |
| kNumTopBits | 24 |
| kTopValue | 2 ^{kNumTopBits} |
| kNumBitModelTotalBits | 11 |
| kBitModelTotal | 2 ^{kNumBitModelTotalBits} |
| kNumMoveBits | 5 |
| kNumBitModelTotalBits | 11 |
| kLenNumLowBits | 3 |
| kLenNumLowSymbols | 2 ^{kLenNumLowBits} |
| kLenNumHighBits | 8 |
| kLenNumHighSymbols | 2 ^{kLenNumHighBits} |
| kNumPosBitsMax | 4 |
| kNumPosStatesMax | 2 ^{kNumPosBitsMax} |
| LenHigh | 2 * (kNumPosStatesMax << kLenNumLowBits) |
| kNumLenProbs | LenHigh + kLenNumHighSymbols |

Table 19 (continued)

| Constant Name | Value |
|-----------------|---------------------|
| kNumAlignBits | 4 |
| kAlignTableSize | $2^{kNumAlignBits}$ |

9.2.5 Process

The initialization process of structures and variables used during the decoding process is shown in [Table 20](#).

Table 20 — Initialization process

```

lzmaParams p
i = 0
d = inputBuffer[i++]
p->prop->lc = d%9
d /= 9
p->prop->pb = d / 5
p->prop->lp = d % 5
p->buf = outputBuffer
p->prop->dictSize =
    ((inputBuffer[i]      ) & 0x000000FF)
  | ((inputBuffer[i+1]<<8 ) & 0x0000FF00)
  | ((inputBuffer[i+2]<<16) & 0x00FF0000)
  | ((inputBuffer[i+3]<<24) & 0xFF000000)
i+= 4

if(p->prop->dictSize < (1<<12))
    p->prop->dictSize = (1<<12)

p->prop->uncompressedSize =
    ((inputBuffer[i]  <<56) & 0xFF00000000000000)
  | ((inputBuffer[i+1]<<48) & 0x00FF000000000000)
  | ((inputBuffer[i+2]<<40) & 0x0000FF0000000000)
  | ((inputBuffer[i+3]<<32) & 0x000000FF00000000)
  | ((inputBuffer[i+4]<<24) & 0x00000000FF000000)
  | ((inputBuffer[i+5]<<16) & 0x0000000000FF0000)
  | ((inputBuffer[i+6]<<8 ) & 0x000000000000FF00)
  | ((inputBuffer[i+7]      ) & 0x00000000000000FF)

i += 8

/* skip one byte */
i++

p->code =
    ((inputBuffer[i]  << 24) & 0xFF000000)
  | ((inputBuffer[i+1]<< 16) & 0x00FF0000)
  | ((inputBuffer[i+2]<< 8 ) & 0x0000FF00)
  | ((inputBuffer[i+3]      ) & 0x000000FF)

p->range = 0xFFFFFFFF

/* with default values numProbs = 8128 */
p->numProbs = kNumBaseProbs +

```

Table 20 (continued)

```

        ( kLzmaLitSize <<(p->prop->lc + p->prop->lp))
for(j = 0; j < p->numProbs; j++)
    p->probs[j] = 1 << 10

p->reps[0] = p->reps[1] = p->reps[2] = p->reps[3] = 1
p->state = 0

LzmaDec_WriteRem(p, p->dicBufSize)
LzmaDec_Decode(p, p->dicBufSize, inputBuffer + i)

```

Table 21 — LzmaDec_WriteRem utility function

```

LzmaDec_WriteRem(p, limit)
{
    if (p->remainLen != 0 && p->remainLen < kMatchSpecLenStart)
    {
        dic = p->dic
        dicPos = p->dicPos
        dicBufSize = p->dicBufSize
        len = p->remainLen
        rep0 = p->reps[0]
        rem = limit - dicPos
        if (rem < len)
            len = rem

        if(p->checkDictSize == 0 && p->prop->dictSize - p->processedPos <= len)
            p->checkDictSize = p->prop->dictSize

        p->processedPos += len
        p->remainLen -= len

        while (len != 0)
        {
            len--
            dic[dicPos]=dic[dicPos - rep0 + (dicPos < rep0 ? dicBufSize : 0)]
            dicPos++
        }
        p->dicPos = dicPos
    }
}

```

9.3 Macros

This subclause defines macros used in the decoding process specified in subclause [9.4](#).

A *macro* is a fragment of code which has been given a name. Whenever the name is used, it is replaced by the contents of the macro. Everything in brackets in the left column is a list of parameters. The names of the parameters passed to the macros are the ones to be used in the replacement.

Table 22 — Macros used in the decoding process

| macro | Code |
|---------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| CALC_POS_STATE(procPos, pbMask) | <code>((procPos) & (pbMask)) << 4)</code> |
| COMBINED_PS_STATE | <code>(posState + state)</code> |
| IF_BIT_0(p) | <pre> ttt = *(p) NORMALIZE bound = (range >> kNumBitModelTotalBits) * ttt if (code < bound) </pre> |
| UPDATE_0(p) | <pre> range = bound *(p) = (CLzmaProb)(ttt + ((kBitModelTotal - ttt) >> kNumMoveBits)) </pre> |
| UPDATE_1(p) | <pre> range -= bound code -= bound *(p) = (CLzmaProb)(ttt - (ttt >> kNumMoveBits)) </pre> |
| GET_BIT2(p, i, A0, A1) | <pre> IF_BIT_0(p) { UPDATE_0(p); i = (i + i); A0; } else { UPDATE_1(p); i = (i + i) + 1; A1; } </pre> |
| TREE_GET_BIT(probs, i) | <code>{GET_BIT2(probs + i, i, , ;); }</code> |
| NORMAL_LITERAL_DEC | <code>TREE_GET_BIT(prob, symbol)</code> |
| MATCHED_LITERAL_DEC | <pre> matchByte += matchByte bit = offs offs &= matchByte probLit = prob + (offs + bit + symbol) GET_BIT2(probLit, symbol, offs ^= bit; , ;) </pre> |
| TREE_6_DECODE(probs, i) | <pre> { i = 1 TREE_GET_BIT(probs, i) TREE_GET_BIT(probs, i) TREE_GET_BIT(probs, i) TREE_GET_BIT(probs, i) TREE_GET_BIT(probs, i) TREE_GET_BIT(probs, i) i -= 0x40 } </pre> |
| REV_BIT(p, i, A0, A1) | <pre> IF_BIT_0(p + i) { UPDATE_0(p + i); A0; } else { UPDATE_1(p + i); A1; } </pre> |
| REV_BIT_VAR(p, i, m) | <code>REV_BIT(p, i, i += m; m += m, m += m; i += m;)</code> |
| REV_BIT_CONST(p, i, m) | <code>REV_BIT(p, i, i += m; , i += m * 2;)</code> |
| REV_BIT_LAST(p, i, m) | <code>REV_BIT(p, i, i -= m , ;)</code> |

Table 22 (continued)

| macro | Code |
|----------------|-----------------------------------------------------------------------------|
| NORMALIZE | if (range < kTopValue) { range <= 8; code = (code << 8) (*buf++); } |
| SpecPos | -kStartOffset |
| IsRep0Long | SpecPos + kNumFullDistances |
| RepLenCoder | IsRep0Long + (kNumStates2 << kNumPosBitsMax) |
| LenCoder | RepLenCoder + kNumLenProbs |
| IsMatch | LenCoder + kNumLenProbs |
| Align | IsMatch + (kNumStates2 << kNumPosBitsMax) |
| IsRep | Align + kAlignTableSize |
| IsRepG0 | IsRep + kNumStates |
| IsRepG1 | IsRepG0 + kNumStates |
| IsRepG2 | IsRepG1 + kNumStates |
| PosSlot | IsRepG2 + kNumStates |
| Literal | PosSlot + (kNumLenToPosStates << kNumPosSlotBits) |
| NUM_BASE_PROBS | Literal + kStartOffset |

9.4 Decoding process

This subclause describes the decoding process of encoded metadata once the initialization specified in subclause 9.2 has initialized the `lzmaProps` and `lzmaParams` structures (see [Tables 14](#) and [15](#)).

Table 23 — LzmaDec_Decode2 function

```

LzmaDec_Decode2(p, limit, buffer[])
{
    do
    {
        limit2 = limit
        if (p->checkDictSize == 0)
        {
            rem = p->prop->dictSize - p->processedPos
            if (limit - p->dicPos > rem)
                limit2 = p->dicPos + rem
        }

        LzmaDec_Decode(p, limit2, buffer)

        if (p->checkDictSize == 0 && p->processedPos >= p->prop->dictSize)
            p->checkDictSize = p->prop->dictSize

        LzmaDec_WriteRem(p, limit);
    }
    while(
        p->dicPos < limit &&
        p->buf < bufLimit &&
        p->remainLen < kMatchSpecLenStart)
}

```

Table 24 — LzmaDec_Decode function implementing the actual LZMA decoding algorithm

```

LzmaDec_Decode(p, limit, buffer[]){

    probs = p->probs_1664
    state = p->state
    rep0 = p->reps[0], rep1 = p->reps[1]
    rep2 = p->reps[2], rep3 = p->reps[3]
    pbMask = (1 << (p->prop->pb)) - 1
    lc = p->prop->lc
    lpMask = (0x100 << p->prop->lp) - (0x100 >> lc)

    dic = p->dic
    dicBufSize = p->dicBufSize
    dicPos = p->dicPos

    processedPos = p->processedPos
    checkDictSize = p->checkDictSize
    len = 0

    buf = p->buf
    range = p->range
    code = p->code
    distance = 0
    posSlot = 0

do
{
    ttt = 0
    bound = 0
    posState = CALC_POS_STATE(processedPos, pbMask)

    prob = probs + IsMatch + COMBINED_PS_STATE
    IF_BIT_0(prob)
    {
        symbol
        UPDATE_0(prob)
        prob = probs + Literal
        if (processedPos != 0 || checkDictSize != 0)

        prob += 3 * (((processedPos << 8) + dic[(dicPos == 0 ?
                                dicBufSize : dicPos) - 1]) & lpMask) << lc)

        processedPos++

        if (state < kNumLitStates)
        {
            state -= (state < 4) ? state : 3
            symbol = 1

            NORMAL_LITER_DEC
            NORMAL_LITER_DEC
            NORMAL_LITER_DEC
            NORMAL_LITER_DEC
            NORMAL_LITER_DEC
        }
    }
}

```

Table 24 (continued)

```

NORMAL_LITER_DEC
NORMAL_LITER_DEC
NORMAL_LITER_DEC
}
else
{
    matchByte=dic[dicPos - rep0
+ (dicPos < rep0 ? dicBufSize : 0)]

    offs = 0x100
    state -= (state < 10) ? 3 : 6
    symbol = 1

    MATCHED_LITER_DEC
    MATCHED_LITER_DEC
    MATCHED_LITER_DEC
    MATCHED_LITER_DEC
    MATCHED_LITER_DEC
    MATCHED_LITER_DEC
    MATCHED_LITER_DEC
    MATCHED_LITER_DEC

}
dic[dicPos++] = symbol & 0xFF
continue
}
{
    UPDATE_1(prob)
    prob = probs + IsRep + state
    IF_BIT_0(prob)
    {
        UPDATE_0(prob)
        state += kNumStates
        prob = probs + LenCoder
    }
    else
    {
        UPDATE_1(prob)
        prob = probs + IsRepG0 + state
        IF_BIT_0(prob)
        {
            UPDATE_0(prob)
            prob = probs + IsRep0Long + COMBINED_PS_STATE
            IF_BIT_0(prob)
            {
                UPDATE_0(prob)
                dic[dicPos]=dic[dicPos-
rep0+(dicPos < rep0 ? dicBufSize : 0)]

                dicPos++
                processedPos++
            }
        }
    }
}

```

Table 24 (continued)

```

state = state <
kNumLitStates ? 9 : 11

        }
        UPDATE_1(prob)
    }
    else
    {
        UPDATE_1(prob)
        prob = probs + IsRepG1 + state
        IF_BIT_0(prob)
        {
            UPDATE_0(prob)
            distance = rep1
        }
        else
        {
            UPDATE_1(prob)
            prob = probs + IsRepG2 + state
            IF_BIT_0(prob)
            {
                UPDATE_0(prob)
                distance = rep2
            }
            else
            {
                UPDATE_1(prob)
                distance = rep3
                rep3 = rep2
            }
            rep2 = rep1
        }
        rep1 = rep0
        rep0 = distance
    }
    state = state < kNumLitStates ? 8 : 11
    prob = probs + RepLenCoder
}
{
    probLen = prob + LenChoice
    IF_BIT_0(probLen)
    {
        UPDATE_0(probLen)
        probLen = prob + LenLow + posState
        len = 1
        TREE_GET_BIT(probLen, len)
        TREE_GET_BIT(probLen, len)
        TREE_GET_BIT(probLen, len)
        len -= 8
    }
}

```

Table 24 (continued)

```

    }
    else
    {
        UPDATE_1(probLen)
        probLen = prob + LenChoice2
        IF_BIT_0(probLen)
        {
            UPDATE_0(probLen)
            probLen=prob+LenLow + posState + (1 << kLenNumLowBits)
            len = 1
            TREE_GET_BIT(probLen, len)
            TREE_GET_BIT(probLen, len)
            TREE_GET_BIT(probLen, len)
        }
        else
        {
            UPDATE_1(probLen)
            probLen = prob + LenHigh
            TREE_DECODE(probLen, (1 << kLenNumHighBits), len)
            len += kLenNumLowSymbols * 2
        }
    }
}

if (state >= kNumStates)
{
    prob=probs + PosSlot +
        ((len < kNumLenToPosStates ? len : kNumLenToPosStates - 1
        << kNumPosSlotBits)

    TREE_6_DECODE(prob, distance)
    if (distance >= kStartPosModelIndex)
    {
        posSlot= distance
        numDirectBits = ((distance >> 1) - 1)
        distance = (2 | (distance & 1))
        if (posSlot < kEndPosModelIndex)
        {
            distance <= numDirectBits
            prob = probs + SpecPos
            {
                m = 1
                distance++
                do
                {
                    REV_BIT_VAR(prob, distance, m)
                }
                while (--numDirectBits)
                distance -= m
            }
        }
    }
}

```

Table 24 (continued)

```

    }
    else
    {
        numDirectBits -= kNumAlignBits
        do
        {
            NORMALIZE
            range >>= 1

            {
                code -= range
                t = (0 - (code >> 31))
                distance = (distance << 1) + (t + 1)
                code += range & t
            }
        }
        while (--numDirectBits)
        prob = probs + Align
        distance <= kNumAlignBits
        {
            i = 1
            REV_BIT_CONST(prob, i, 1)
            REV_BIT_CONST(prob, i, 2)
            REV_BIT_CONST(prob, i, 4)
            REV_BIT_LAST (prob, i, 8)
            distance |= i
        }
        if (distance == 0xFFFFFFFF)
        {
            len = kMatchSpecLenStart
            state -= kNumStates
            break
        }
    }
}
rep3 = rep2
rep2 = rep1
rep1 = rep0
rep0 = distance + 1
state = (state < kNumStates + kNumLitStates) ? kNumLitStates :
kNumLitStates + 3
if (distance >= (checkDictSize == 0 ? processedPos:
checkDictSize))
{
    p->dicPos = dicPos
    return_error()
}
}

len += kMatchMinLen

```


Table 24 (continued)

```

{
    rem = 0
    curLen = 0
    pos = 0

    if ((rem = limit - dicPos) == 0)
    {
        p->dicPos = dicPos
        return_error()
    }

    curLen = ((rem < len) ? rem : len)
    pos = dicPos - rep0 + (dicPos < rep0 ? dicBufSize : 0)
    processedPos += curLen

    len -= curLen
    if (curLen <= dicBufSize - pos)
    {
        dest[] = dic + dicPos
        src = pos - dicPos
        dicPos += curLen

        do
            dest[k] = dest[k + src]
            while (++k != curLen)

        dest += curLen
    }
    else
    {
        do
        {
            dic[dicPos++] = dic[pos]
            if (++pos == dicBufSize)
                pos = 0
        }
        while (--curLen != 0)
    }
}

while (dicPos < limit && buf < bufLimit)

NORMALIZE

/* copy the decoded bytes in the output buffer */
p->buf = buf
p->range = range
p->code = code
p->remainLen = len
p->dicPos = dicPos
p->processedPos = processedPos

```

Table 24 (continued)

```

p->reps[0] = rep0
p->reps[1] = rep1
p->reps[2] = rep2
p->reps[3] = rep3
p->state = state
}

```

10 Application programming interfaces (APIs)

10.1 General

This clause contains the application programming interface (API) to coded genomic information compliant with the ISO/IEC 23092 series. It specifies the interfaces to be available in a tool implementing the API. The API may be implemented locally or remotely.

In case the information is protected, operations are controlled by privacy rules, as specified in [Clause 7](#). Whenever the caller of functions is not authorized to access the full content, only the content for which the caller is authorized is returned.

10.2 Structure of the API

The API specifies interfaces that handle data structures, as specified in ISO/IEC 23092-1, ISO/IEC 23092-2 and this document. Operations are applied to data structures that are organized in hierarchy levels. In the context of this document, a hierarchy level defines the scope of an operation. The hierarchy levels considered are dataset group, dataset and access unit, as specified in ISO/IEC 23092-1 and ISO/IEC 23092-2.

The context of the API is one bitstream (either in File or Transport format), as specified in ISO/IEC 23092-1. How the API gets access to the file or bitstream is left open to implementation.

[Table 25](#) lists the specified functions groups.

Table 25 — Groups of functions

| Functions group | Brief description |
|---------------------|-------------------------------------------------------------------------------------------------------------------------------------------|
| Genomic information | Functions used to query the structure of, and retrieve, the genomic information coded in a bitstream compliant with ISO/IEC 23092 series. |
| Metadata | Functions used to query the structure of, and retrieve, the metadata associated with the coded genomic data. |
| Protection | Functions used to retrieve the protection metadata associated with the coded genomic data. |
| Reference | Functions used to retrieve the reference associated with a dataset. |
| Statistics | Functions used to retrieve statistics associated with a dataset. |

10.3 Detailed specification of the API

10.3.1 Data types

The data types used in this document are defined in [Table 26](#).

Table 26 — Data types

| | |
|--------------|----------------------------------------------------|
| uint | unsigned integer number |
| int | signed integer number |
| float | floating point number, as defined in IEEE 754-2008 |
| char | printable ASCII character |
| st(v) | string, as specified in subclause 5.2 |
| bool | Boolean value |
| mpegRecord | MPEG-G record, as specified in ISO/IEC 23092-2 |
| rawReference | Raw reference, as specified in ISO/IEC 23092-2 |

10.3.2 Return codes

Table 27 specifies the return codes returned by calls to the operations specified in subclauses 10.3.6 to 10.3.10.

Error codes are of type `returnCodeT`. Return code values are listed in Table 27.

Table 27 — Return codes

| Name | Value | Description |
|---------------------------|-----------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| G_SUCCESS | 0 | The operation completed successfully. This includes the case when the access is authorized but the output is empty because no content matches the input parameters. |
| G_PARTIALLY_AUTHORIZED | 1 | The operation completes successfully, but only part of the queried content is returned due to partial lack of authorization. |
| G_NOT_AUTHORIZED | 2 | The operation is not authorized according the privacy rules. |
| G_VERIFICATION_FAILED | 3 | The digital signature, as specified in subclause 7.4, is not verified. |
| G_DECRYPTION_FAILED | 4 | It is not possible to complete the decryption process. |
| G_DATASETGROUP_NOTFOUND | 5 | The requested dataset group does not exist. |
| G_DATASET_NOTFOUND | 6 | The requested dataset does not exist. |
| G_ACCESSUNITS_NOTFOUND | 7 | The requested access unit does not exist. |
| G_REFERENCE_NOTFOUND | 8 | The requested reference does not exist. |
| G_SEQUENCE_NOTFOUND | 9 | The requested sequence does not exist. |
| G_METADATA_FIELD_NOTFOUND | 10 | The metadata element does not exist. If the element exists but is empty, this error is not returned. |
| G_INVALID_METADATA | 11 | The value of the metadata element, or the element itself, does not comply with the schema. |
| G_INVALID_REFERENCE | 12 | This error is returned when the genomic reference to be used in the decoding process is not compatible with the genomic information. |
| G_INVALID_PARAMETER | 13 | At least one parameter to the call is invalid. |
| G_INVALID_BITSTREAM | 14 | The provided bitstream is invalid. |
| G_UNLISTED_ERROR | 15 | An unlisted error. |
| | 16 .. 255 | Reserved |

10.3.3 Metadata fields

Metadata fields are represented as variables of type `st(v)`. The content of a metadata field is a valid W3C XPath 1.0 path. The metadata field expression shall return only one node.

10.3.4 Output structures

10.3.4.1 General

This subclause specifies the syntax of the output structures used to convey the results returned by calls to the functions specified in this document.

10.3.4.2 Hierarchy

This subclause specifies the syntax of the output structure used to convey the results returned by calls to the GetHierarchy function.

```
struct outHierarchyT {
    uint    datasetGroupsCount;
    uint    datasetGroupID[];
    uint    datasetsCount[];
    uint    datasetID[][];
}
```

Each `outHierarchyT` associates a dataset group with all the datasets belonging to it.

Members of the `outHierarchyT` structure

| | |
|---------------------------------|---------------------------------------------------------------------------|
| <code>datasetGroupsCount</code> | the number of dataset groups present in the bitstream |
| <code>datasetGroupID</code> | list of identifiers of dataset groups present in the bitstream |
| <code>datasetsCount</code> | list containing the number of datasets associated with each dataset group |
| <code>datasetID</code> | list of identifiers of datasets associated with each dataset group |

10.3.4.3 Records

This subclause specifies the syntax of the output structure used to convey the results returned by calls to the GetDataByLabel function.

```
struct outRecordsT {
    uint    datasetGroupID;
    uint    datasetID;
    uint    recordsCount;
    mpegRecord records[]
    genAuxRecord auxInfo[];
}
```

Members of the `outRecordsT` structure

| | |
|-----------------------------|------------------------------------------------------------------------------------------------------|
| <code>datasetGroupID</code> | identifier of a dataset group |
| <code>datasetID</code> | identifier of a dataset associated with the dataset group identified by <code>datasetGroupID</code> |
| <code>recordsCount</code> | number of records returned by the call |
| <code>records[]</code> | list of records returned by the call |
| <code>auxInfo[]</code> | list of returned <code>genAuxRecord</code> structures associated to the records returned by the call |

10.3.4.4 References

This subclause specifies the syntax of the output structure used to convey the results returned by calls to the GetDatasetReference function.

```
struct outReferenceT {
    uint    seqCount;
```

```

    st(v) seqName[];
    uint  seqStart[];
    uint  seqEnd[];
    char  refSequence[][];
}

```

Members of the `outReferenceT` structure

| | |
|-------------|-------------------------------------------------------------------------------------|
| seqCount | number of reference sequences returned by the call |
| seqName | list containing the names of the reference sequences returned by the call |
| seqStart | list containing the start position of the reference sequences returned by the call |
| seqEnd | list containing the end position of the reference sequences returned by the call |
| refSequence | lists of ASCII characters representing the reference sequences returned by the call |

10.3.4.5 Region protection

This subclause specifies the syntax of the output structure used to convey the results returned by calls to the `GetRegionProtection` function.

```

struct outRegionProtectionT {
    string  sequenceName;
    uint    startPos;
    uint    endPos;
    uint    classID;
    uint    numKeys
    st(v)   keyName[numKeys]
}

```

Members of the `outRegionProtectionT` structure

| | |
|--------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| sequenceName | name of the sequence where the protected region is defined |
| startPos | start position of the protected region. |
| endPos | end position of the protected region. |
| classID | the class of access units being encrypted as here specified |
| numKeys | the number of keys |
| keyName | any key name present in this array allows to decrypt the content of an access unit which range intersects with the region defined by <code>sequenceName</code> , <code>startPos</code> , <code>endPos</code> , <code>classID</code> |

10.3.4.6 Simple statistics

This subclause specifies the syntax of the output structure used to convey the results returned by calls to the `GetSimpleStatistics` function.

```

struct simpleSegmentStatisticsT {
    uint    readsNumber;
    uint    segmentsNumberReadsDistribution[maxSegments];
    uint    qualityCheckFailedReadsNumber;
    uint    segmentLength[3];
    uint    mappedStrandSegmentDistribution[3];
    uint    properlyPairedNumber;
    uint    mappedStrandPairDistribution[9];
    uint    maxAlignments;
    uint    multipleAlignmentSegmentDistribution[maxAlignments + 1];
    uint    coverage[3];
    uint    weightedCoverage[3];
    uint    errorsNumber[3];
}

```

```

uint substitutionsNumber[3];
uint insertionsNumber[3];
uint insertionsLength[3];
uint deletionsNumber[3];
uint deletionsLength[3];
uint splicesNumber[3];
uint splicesLength[3];
uint alignmentScore[3];
uint classSegmentDistribution[6];
uint clippedSegmentDistribution[3];
uint opticalDuplicatesNumber;
uint chimerasNumber;
}

```

| Name | Semantics |
|---------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| readsNumber | Total number of the reads to which the selected segments belong. |
| segmentsNumberReadsDistribution | Distribution by segment number of the reads to which the selected segments belong (one per number of segments possible; single end reads are stored at position 0, paired end reads at position 1, and so on). <code>maxSegments</code> as specified in 10.3.10.1 . |
| qualityCheckFailedReadsNumber | Number of reads flagged as “failing vendor quality checks” while considering the set of reads to which the selected segments belong. |
| segmentLength | Array of three numbers containing minimum, maximum and average lengths (in this order) for the selected segments. Average rounded to the closest integer. |
| mappedStrandSegmentDistribution | Array of three numbers containing: pos 0: Number of unmapped segments pos 1: Number of segments mapped to the forward strand pos 2: Number of segments mapped to the reverse strand for all selected segments. |
| properlyPairedNumber | Number of reads flagged as being properly paired (paired-end, both pairs aligned within proper distance) while considering the set of reads to which the selected segments belong. |
| mappedStrandPairDistribution | Distribution of paired-end reads according to the following classification: Position 0: reads having first and second segments unmapped Position 1: reads having first segment unmapped and first best alignment for the second segment on the forward strand Position 2: reads having first segment unmapped and first best alignment for the second segment on the reverse strand Position 3: reads having first best alignment for the first segment on the forward strand and second segment unmapped Position 4: reads having first best alignment for the first segment on the reverse strand and the second segment unmapped Position 5: reads having first best alignment for the first segment on the forward strand and first best alignment for the second strand on the forward strand |

| Name | Semantics |
|--------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| | <p>Position 6: reads having first best alignment for the first segment on the forward strand and first best alignment for the second strand on the reverse strand</p> <p>Position 7: reads having first best alignment for the first segment on the reverse strand and first best alignment for the second strand on the forward strand</p> <p>Position 8: reads having first best alignment for the first segment on the reverse strand and first best alignment for the second strand on the reverse strand when considering the set of paired-end reads to which the selected segments belong.</p> |
| maxAlignments | Maximum number of alignments while considering the set of reads to which the selected segments belong. |
| multipleAlignmentSegmentDistribution | Distribution of the number of multiple alignments for the selected segments. Position n in the array corresponds to the segments having n alignments. |
| coverage | Array of three numbers containing minimum, maximum and average coverage (in this order) across the range considered, taking into account the first best alignment of the selected segments. Average rounded to the closest integer. |
| weightedCoverage | Array of three numbers containing minimum, maximum and average weighted coverage (in this order) across the range considered, taking into account all n alignments of the selected segments having an alignment score equal to the maximum one, and weighting each alignment by 1/n. Average rounded to the closest integer. |
| errorsNumber | Array of three numbers containing minimum, maximum and average number (in this order) of errors (sum of substitutions, insertions and deletions) present in the extended CIGAR for the first best alignment of the selected segments. Average rounded to the closest integer. |
| substitutionsNumber | Array of three numbers containing minimum, maximum and average number (in this order) of substitutions present in the extended CIGAR for the first best alignment of the selected segments. Average rounded to the closest integer. |
| insertionsNumber | Array of three numbers containing minimum, maximum and average number (in this order) of insertion operations present in the extended CIGAR for the first best alignment of the selected segments. Average rounded to the closest integer. |
| insertionsLength | Array of three numbers containing minimum, maximum and average cumulative length (in this order) of the insertion operations present in the extended CIGAR for the first best alignment of the selected segments. Average rounded to the closest integer. |
| deletionsNumber | Array of three numbers containing minimum, maximum and average number (in this order) of deletion operations present in the extended CIGAR for the first best alignment of the selected segments. Average rounded to the closest integer. |
| deletionsLength | Array of three numbers containing minimum, maximum and average cumulative length (in this order) of deletion operations present in the extended CIGAR for the first best alignment of the selected segments. Average rounded to the closest integer. |

| Name | Semantics |
|----------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| splicesNumber | Array of three numbers containing minimum, maximum and average number (in this order) of splice operations present in the extended CIGAR for the first best alignment of the selected segments. Average rounded to the closest integer. |
| splicesLength | Array of three numbers containing minimum, maximum and average cumulative length (in this order) of splice operations present in the extended CIGAR for the first best alignment of the selected segments. Average rounded to the closest integer. |
| alignmentScore | Array of three numbers containing minimum, maximum and average (in this order) of the alignment score for the first best alignment of the selected segments. Average rounded to the closest integer. |
| classSegmentDistribution | Distribution of segments according to the classification defined in ISO/IEC 23092-2, Index is the class_id specified in subclause 5.4 minus 1. |
| clippedSegmentDistribution | Distribution of segments according to the following classification pos 0: number of segments without clipped bases pos 1: number of segments with soft-clipped bases and without hard-clipped bases pos 2: number of segments with hard-and soft-clipped bases. |
| opticalDuplicatesNumber | Number of reads flagged as being optical duplicates while considering the set of reads to which the selected segments belong. |
| chimerasNumber | Number of reads flagged as being chimeras (can be aligned only taking into account more than one reference sequence) while considering the set of reads to which the selected segments belong. |

10.3.4.7 Advanced statistics

This subclause specifies the syntax of the output structure used to convey the results returned by calls to the GetAdvancedStatistics function.

```

struct extendedSegmentStatisticsT {
    uint    mappedBasesNumberDistribution[segmentLength[1] + 1];
    uint    coverage[endPos - startPos + 1];
    uint    weightedCoverage[endPos - startPos + 1];
    uint    errorsNumberDistribution[errorsNumber[1] + 1];
    uint    errorsPositionDistribution[segmentLength[1]];
    uint    substitutionsNumberDistribution[substitutionsNumber[1] + 1];
    uint    substitutionsTransitionDistribution[5][5][segmentLength[1]];
    uint    substitutionsPositionDistribution[segmentLength[1]];
    uint    insertionsNumberDistribution[insertionsNumber[1] + 1];
    uint    insertionsLengthDistribution[insertionsLength[1] + 1];
    uint    insertionsPositionDistribution[segmentLength[1]];
    uint    deletionsNumberDistribution[deletionsNumber[1] + 1];
    uint    deletionsLengthDistribution[deletionsLength[1] + 1];
    uint    deletionsPositionDistribution[segmentLength[1]];
    uint    splicesNumberDistribution[3][splicesNumber[1] + 1];
    uint    splicesLengthDistribution[3][splicesLength[1] + 1];
    uint    splicesPositionDistribution[3][segmentLength[1]];
    uint    alignmentScoreValueDistribution[alignmentScore[1] + 1];
    uint    alignmentScoreSegmentLengthDistribution[3][segmentLength[1]];
    uint    basesPositionDistribution[5][segmentLength[1]];
    uint    gcContentValueDistribution[101];
    uint    maxQualityScore;
    uint    qualityScoreDistribution[maxQualityScore + 1];
    uint    qualityScorePositionDistribution[3][segmentLength[1]];

```

```

uint  qualityScorePositionPercentilesDistribution[6][segmentLength[1]];
}

```

| Name | Semantics |
|-------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| mappedBasesNumberDistribution | Distribution of segments by their number of mapped bases. A base is considered mapped whenever it is exactly matched (no substitution or indel cover it) in the first best alignment. The index ranges from zero (no bases mapped) to <code>segmentLength[1]</code> (all bases mapped). |
| coverage | Array of <code>endPos - startPos + 1</code> the coverage (number of alignments without insertions or deletions covering each base) across the requested range, taking into account the first best alignment of the selected segments. |
| weightedCoverage | Array of <code>endPos - startPos + 1</code> numbers containing the coverage (number of alignments without insertions or deletions covering each base) across the requested range, taking into account all <code>n</code> alignments of the selected segments having an alignment score equal to the maximum one, and weighting each alignment by $1/n$. Coverage rounded to the closest integer. |
| errorsNumberDistribution | Distribution of segments by their number of errors (the sum of the number of substitutions, insertions, and deletions) in their first best alignments. The index ranges from zero (no error) to <code>errorsNumber[1]</code> . |
| errorsPositionDistribution | Array of dimension <code>segmentLength[1]</code> . To position <code>n</code> it corresponds the cumulative number of segments for which an error occurs at position <code>n</code> in the first best alignment. |
| substitutionsNumberDistribution | Distribution of segments by their number of substitutions in their first best alignments. The index ranges from zero (no substitution) to <code>substitutionsNumber[1]</code> . |
| substitutionsTransitionDistribution | Array of dimension $5 * 5 * \text{length}[1]$. Defining the vector V as $V = \{A, C, G, N, T\}$, <code>substitutionsTransitionDistribution[i][j][n]</code> represents the cumulative number of substitutions of the symbol $V[i]$ with symbol $V[j]$ occurring at position <code>n</code> in the first best alignments for all the segments. |
| substitutionsPositionDistribution | Array of dimension <code>segmentLength[1]</code> . To position <code>n</code> it corresponds the cumulative number of segments for which a substitution occurs at position <code>n</code> in the first best alignment. |
| insertionsNumberDistribution | Distribution of segments by their number of insertions in their first best alignments. The index ranges from zero (no insertion) to <code>insertionsNumber[1]</code> . |
| insertionsLengthDistribution | Distribution of segments by the cumulative length of insertion operations in their first best alignments. The index ranges from zero (no insertion) to <code>insertionsLength[1]</code> . |
| insertionsPositionDistribution | Array of dimension <code>segmentLength[1]</code> . To position <code>n</code> it corresponds the cumulative number of segments for which an insertion operation starts at position <code>n</code> in the first best alignment. |

| Name | Semantics |
|-----------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| deletionsNumberDistribution | Distribution of segments by their number of deletions in their first best alignments. The index ranges from zero (no deletion) to <code>deletionsNumber[1]</code> . |
| deletionsLengthDistribution | Distribution of segments by the cumulative length of deletion operations in their first best alignments. The index ranges from zero (no deletion) to <code>deletionsLength[1]</code> . |
| deletionsPositionDistribution | Array of dimension <code>segmentLength[1]</code> . To position <code>n</code> it corresponds the cumulative number of segments for which a deletion operation starts at position <code>n</code> in the first best alignment. |
| splicesNumberDistribution | Distribution of segments by the physical strand of the molecule originating the segment, and the number of splices in their first best alignments. The first index is 0 when the strand of the original RNA molecule is not known, 1 when the strand is forward and 2 when the strand is reverse. The second index ranges from zero (no splice) to <code>splicesNumber[1]</code> . |
| splicesLengthDistribution | Distribution of segments by the physical strand of the molecule originating the segment, and the cumulative length of splice operations in their first best alignments. The first index is 0 when the strand of the original RNA molecule is not known, 1 when the strand is forward and 2 when the strand is reverse. The second index ranges from zero (no splice) to <code>splicesLength[1]</code> . |
| splicesPositionDistribution | Array of dimension $3 * \text{segmentLength}[1]$. The first index is 0 when the strand of the original RNA molecule is not known, 1 when the strand is forward and 2 when the strand is reverse. To a value <code>n</code> for the second index it corresponds the cumulative number of segments for which a splice operation starts at position <code>n</code> in the first best alignment. |
| alignmentScoreValueDistribution | Distribution of segments by the value of the score of their first best alignment. The index ranges from zero to <code>alignmentScore[1]</code> . Only applicable if scores are coded as integer numbers as specified in ISO/IEC 23092-2. |
| alignmentScoreSegmentLengthDistribution | Bi-dimensional array where the first index <code>i</code> of <code>alignmentScoreSegmentLengthDistribution[i][j]</code> corresponds to minimum, maximum and average (in this order) of the alignment score, taking into account the first best alignment of the selected segments having <code>segmentLength</code> equal to <code>j</code> . Average rounded to the closest integer. Only applicable if scores are coded as integer numbers as specified in ISO/IEC 23092-2. |
| basesPositionDistribution | Bi-dimensional array. Defining the vector <code>V</code> as <code>V = {A, C, G, N, T}</code> , <code>basesPositionDistribution[i][j]</code> corresponds to the cumulative number of segments having nucleotide <code>V[i]</code> at position <code>j</code> . |
| gcContent | Distribution of binned fractional GC read content: <code>gc_content[i]</code> counts the number of segments having a GC content $\geq i/100$ and $< (i+1)/100$ (e.g. bin 10 counts the reads having at least 10 %, and less than 11%, GC content). |

| Name | Semantics |
|---------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| maxQualityScore | Maximum Phred quality score recorded among those for all the nucleotides of the selected segments. |
| qualityScoreDistribution | Distribution of nucleotides by their Phred quality score across all the selected segments. The index ranges from zero to maxQualityScore. |
| qualityScorePositionDistribution | Bi-dimensional array where the first index <i>i</i> of qualityScorePositionDistribution [<i>i</i>][<i>j</i>] corresponds to minimum, maximum and average (in this order) of the Phred quality scores for all the nucleotides at position <i>j</i> , calculated across all the selected segments. Average rounded to the closest integer. |
| qualityScorePositionPercentilesDistribution | Bi-dimensional array where the first index <i>i</i> of qualityScorePositionPercentilesDistribution [<i>i</i>][<i>j</i>] corresponds to the number of nucleotides having Phred quality scores in the ranges 0-10 %, 10-25 %, 25-50 %, 50-75 %, 75-90 %, 90-100 % (in this order; bottom interval range included and top interval range excluded except in the case of the last bin) at position <i>j</i> of the segment, for all the selected segments. |

```

struct advancedSegmentStatisticsT {
    simpleSegmentStatisticsT simpleStatistics[2];
    extendedSegmentStatisticsT extendedStatistics[2];
}

```

| Name | Semantics |
|--------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| simpleStatistics | Array of two simpleSegmentStatisticsT structures containing statistics on the segments as specified in subclause 10.3.4.6. The first structure (index 0) contains statistics for the segments having the first best alignment on the forward strand of the reference, the second structure (index 1) contains statistics for the segments having first best alignment on the reverse strand of the reference. |
| extendedStatistics | Array of two extendedSegmentStatisticsT structures containing statistics on the segments as specified in this clause. The first structure (index 0) contains statistics for the segments having the first best alignment on the forward strand of the reference, the second structure (index 1) contains statistics for the segments having first best alignment on the reverse strand of the reference. |

10.3.5 Filters

10.3.5.1 General

This subclause specifies data structures used to express criteria to select MPEG-G records to be retrieved.

10.3.5.2 Simple filter

This subclause specifies a data structure to apply simple filtering on MPEG-G records.

```

struct simpleFilterT {
    uint    numberOfGroups;
    st(v)  groupNames[numberOfGroups];
    bool    classID[6];
    uint    sequenceID;
}

```

```

uint  startPos;
uint  endPos;
bool  singleEndsStrand[3];
bool  pairedEndsStrand[9];
bool  includeClippedReads[2];
bool  includeMultipleAlignments;
bool  includeOpticalDuplicates;
bool  includeQualityCheckFailed;
bool  includeAuxRecords;
bool  includeReadNames;
bool  includeQualityValues;
bool  mismatchesIncludeNs;
}

```

| Name | Semantics |
|---------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| numberOfGroups | Number of read groups for which records are to be retrieved. 0 means all groups. |
| groupNames | Array of read group names for which records are to be retrieved. |
| classID | Array of Boolean specifying the class of records (as defined in ISO/IEC 23092-2:2019, subclause 5.4) to be considered when applying the filter. When <code>classID[i]</code> is set to true records having <code>class_id i+1</code> shall be selected. |
| sequenceID | Identifier of the reference sequence to be considered when filtering. |
| startPos | Start position of the queried region. It shall be less than or equal to <code>endPos</code> otherwise a <code>G_INVALID_PARAMETER</code> error is returned. Inclusive. |
| endPos | End position of the queried region. It shall be greater than or equal to <code>startPos</code> otherwise a <code>G_INVALID_PARAMETER</code> error is returned. Inclusive. |
| singleEndsStrand | <code>singleEndsStrand</code> is an array of 3 Boolean flags specifying whether single-end records being unmapped (position 0), having first best alignment to the forward strand (position 1), and having first best alignment to the reverse strand (position 2) should be selected. More than one flag can be set. |
| pairedEndsStrand | <code>pairedEndsStrand</code> is an array of 9 Boolean flags specifying whether paired-end records having first and second segment unmapped (position 0), having first segment unmapped and first best alignment for the second segment on the forward strand (position 1), having first segment unmapped and first best alignment for the second segment on the reverse strand (position 2), having first best alignment for the first segment on the forward strand and second segment unmapped (position 3), having first best alignment for the first segment on the reverse strand and the second segment unmapped (position 4), having first best alignment for the first segment on the forward strand and first best alignment for the second strand on the forward strand (position 5), having first best alignment for the first segment on the forward strand and first best alignment for the second strand on the reverse strand (position 6), having first best alignment for the first segment on the reverse strand and first best alignment for the second strand on the forward strand (position 7), having first best alignment for the first segment on the reverse strand and first best alignment for the second strand on the reverse strand (position 8), should be selected. More than one flag can be set. |
| includeClippedReads | Array of Boolean fields used to select the alignments to be retrieved according to the following criteria bit 1 set: if set to true, include in the result reads with soft-clipped bases bit 2 set: if set to true, include in the result reads with hard-clipped bases |
| includeMultipleAlignments | When set to true, select records with multiple alignments (if the mpeg record has more alignments it is considered to have more alignments). |

| Name | Semantics |
|---------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| includeOpticalDuplicates | When set to true, select records with the optical duplicate flag set as specified in ISO/IEC 23092-2. |
| includeQualityCheckFailed | When set to true, select records with the quality check failed flag set as specified in ISO/IEC 23092-2. |
| includeAuxRecords | When set to true, include coded auxiliary fields carried by the AU_information_value array contained in the gen_info element AU_information defined in ISO/IEC 23092-1, as described in Clause 8 . |
| includeReadNames | When set to true, include read names if available. |
| includeQualityValues | When set to true, include quality values if available. |
| mismatchesIncludeNs | Consider Ns as mismatches. When set to false class N records are considered equivalent to class P records. |

10.3.5.3 Advanced filter

This subclause specifies selective access criteria for segments of coded reads.

```

struct segmentFilterT{
    bool    filterScope[3];
    uint    mappedBasesRange[2];
    float   mappedFractionRange[2];
    uint    errorsRange[2];
    float   errorsFractionRange[2];
    uint    substitutionsRange[2];
    float   substitutionsFractionRange[2];
    uint    insertionsRange[2];
    float   insertionsFractionRange[2];
    uint    insertionsLengthRange[2];
    float   insertionsLengthFractionRange[2];
    uint    deletionsRange[2];
    float   deletionsFractionRange[2];
    uint    deletionsLengthRange[2];
    float   deletionsLengthFractionRange[2];
    uint    splicesRange[2];
    float   splicesFractionRange[2];
    uint    splicesLengthRange[2];
    int     splicesDirectionAsEnd;
    float   alignmentScoreRange[2];
    uint    qualityScoreRange[2];
}

```

| Name | Semantics |
|------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| filterScope | Array of Boolean fields used to select the segments to be retrieved according to the following criteria: bit 1 set: single-end read bit 2 set: paired-end first read bit 3 set: paired-end second read |
| mappedBasesRange | mappedBasesRange[0] is the minimum number of mapped bases in the first best alignment for the segment in order for the segment to be selected (inclusive). mappedBasesRange[1] is the maximum number of mapped bases in the first best alignment for the segment in order for a segment to be selected (inclusive). |

| Name | Semantics |
|----------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| mappedFractionRange | <p>mappedFractionRange[0] is the minimum fraction: mapped bases in the first best alignment for the segment divided by the number of bases in the segment in order for the segment to be selected (inclusive).</p> <p>mappedFractionRange[1] is the maximum fraction: mapped bases in the first best alignment for the segment divided by the number of bases in the segment in order for the segment to be selected (inclusive).</p> |
| errorsRange | <p>errorsRange[0] is the minimum number of errors (sum of substitutions, insertions and deletions) in the first best alignment for the segment in order for the segment to be selected (inclusive).</p> <p>errorsRange[1] is the maximum number of errors (sum of substitutions, insertions and deletions) in the first best alignment for the segment in order for the segment to be selected (inclusive).</p> |
| errorsFractionRange | <p>errorsFractionRange[0] is the minimum fraction of errors (sum of substitutions, insertions and deletions) in the first best alignment for the segment in order for the segment to be selected (inclusive).</p> <p>errorsFractionRange[1] is the maximum fraction of errors (sum of substitutions, insertions and deletions) in the first best alignment for the segment in order for the segment to be selected (inclusive).</p> |
| substitutionsRange | <p>substitutionsRange[0] is the minimum number of substitutions in the first best alignment for the segment in order for the segment to be selected (inclusive).</p> <p>substitutionsRange[1] is the maximum number of substitutions in the first best alignment for the segment in order for the segment to be selected (inclusive).</p> |
| substitutionsFractionRange | <p>substitutionsFractionRange[0] is the minimum fraction of substitutions in the first best alignment for the segment in order for the segment to be selected (inclusive).</p> <p>substitutionsFractionRange[1] is the maximum fraction of substitutions in the first best alignment for the segment in order for the segment to be selected (inclusive).</p> |
| insertionsRange | <p>insertionsRange[0] is the minimum number of insertions in the first best alignment for the segment in order for the segment to be selected (inclusive).</p> <p>insertionsRange[1] is the maximum number of insertions in the first best alignment for the segment in order for the segment to be selected (inclusive).</p> |
| insertionsFractionRange | <p>insertionsFractionRange[0] is the minimum fraction of insertions in the first best alignment for the segment in order for the segment to be selected (inclusive).</p> <p>insertionsFractionRange[1] is the maximum fraction of insertions in the first best alignment for the segment in order for the segment to be selected (inclusive).</p> |
| insertionsLengthRange | <p>insertionsLengthRange[0] is the minimum length of insertions in the first best alignment for the segment in order for the segment to be selected (inclusive).</p> <p>insertionsLengthRange[1] is the maximum length of insertions in the first best alignment for the segment in order for the segment to be selected (inclusive).</p> |

| Name | Semantics |
|-------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| insertionsLengthFractionRange | <p>insertionsLengthFractionRange[0] is the minimum fraction of insertion length in the first best alignment for the segment in order for the segment to be selected (inclusive).</p> <p>insertionsLengthFractionRange[1] is the maximum fraction of insertion length in the first best alignment for the segment in order for the segment to be selected (inclusive).</p> |
| deletionsRange | <p>deletionsRange[0] is the minimum number of insertions in the first best alignment for the segment in order for the segment to be selected (inclusive).</p> <p>deletionsRange[1] is the maximum number of insertions in the first best alignment for the segment in order for the segment to be selected (inclusive).</p> |
| deletionsFractionRange | <p>deletionsFractionRange[0] is the minimum fraction of insertions in the first best alignment for the segment in order for the segment to be selected (inclusive).</p> <p>deletionsFractionRange[1] is the maximum fraction of insertions in the first best alignment for the segment in order for the segment to be selected (inclusive).</p> |
| deletionsLengthRange | <p>deletionsLengthRange[0] is the minimum length of insertions in the first best alignment for the segment in order for the segment to be selected (inclusive).</p> <p>deletionsLengthRange[1] is the maximum length of insertions in the first best alignment for the segment in order for the segment to be selected (inclusive).</p> |
| deletionsLengthFractionRange | <p>deletionsLengthFractionRange[0] is the minimum fraction of insertion length in the first best alignment for the segment in order for the segment to be selected (inclusive).</p> <p>deletionsLengthFractionRange[1] is the maximum fraction of insertion length in the first best alignment for the segment in order for the segment to be selected (inclusive).</p> |
| splicesRange | <p>splicesRange[0] is the minimum number of splices in the first best alignment for the segment in order for the segment to be selected (inclusive).</p> <p>splicesRange[1] is the maximum number of splices in the first best alignment for the segment in order for the segment to be selected (inclusive).</p> |
| splicesFractionRange | <p>splicesFractionRange[0] is the minimum fraction of splices in the first best alignment for the segment in order for the segment to be selected (inclusive).</p> <p>splicesFractionRange[1] is the maximum fraction of splices in the first best alignment for the segment in order for the segment to be selected (inclusive).</p> |
| splicesLengthRange | <p>splicesLengthRange[0] is the minimum length of splices in the first best alignment for the segment in order for the segment to be selected (inclusive).</p> <p>splicesLengthRange[1] is the maximum length of splices in the first best alignment for the segment in order for the segment to be selected (inclusive).</p> |
| splicesDirectionAsEnd | <p>If set to n, only select records for which the physical strand of the original RNA molecule is the same as, or opposite to, the strand to which segment n is aligned (first best alignment considered). If sign of n is positive, strand is the same; if sign of n is negative, strand is opposite. For instance, a value of -2 would only consider records for which the first best alignment for the second segment is on the opposite strand to the strand the RNA molecule originally comes from.</p> |

| Name | Semantics |
|---------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| alignmentScoreRange | alignmentScoreRange[0] is the minimum score of the first best alignment for the segment in order for the segment to be selected (inclusive). alignmentScoreRange[1] is the maximum score of the first best alignment for the segment in order for the segment to be selected (inclusive). |
| qualityScoreRange | qualityScoreRange[0] is the minimum Phred base quality score for a segment to be selected. qualityScoreRange[1] is the maximum Phred base quality score for a segment to be selected. |

```

struct advancedFilterT {
    simpleFilterT filter;
    uint segmentFiltersCount;
    segmentFilterT segmentFilters[segmentFiltersCount];
}

```

| Name | Semantics |
|---------------------|-----------------------------------------------------------------|
| Filter | A filter specified as in subclause 10.3.5.2 . |
| segmentFiltersCount | Number of filters to be applied at a segment level. |
| segmentFilters | Array of segmentFilterT structures as specified in this clause. |

10.3.6 Genomic information

10.3.6.1 General

This subclause specifies API function calls returning genomic information as collections of MPEG-G records.

10.3.6.2 GetHierarchy

Signature

```

returnCodeT GetHierarchy (
    [out] outHierarchyT outputHierarchy
);

```

Description

This function returns an array of outHierarchyT structures that describes the content of a file or bitstream compliant with ISO/IEC 23092 series.

Parameters

| Name | Description |
|-----------------|---------------------------------------------------------------------------------------------------|
| outputHierarchy | the outHierarchyT structure describing all the dataset groups contained in the file or bitstream. |

Authorization

The getHierarchy call returns only information for which the caller has all necessary keys to decrypt the information.

Return values

A call to this operation shall return one of these error codes: G_SUCCESS, G_NOT_AUTHORIZED, G_INVALID_BITSTREAM.

10.3.6.3 GetDataBySimpleFilter

Signature

```
returnCodeT GetDataBySimpleFilter (
    [in] uintdatasetGroupID,
    [in] uintdatasetID,
    [in] simpleFilterT filter,
    [out] outRecordsT outputRecords
);
```

Description

This function returns an `outRecordsT` structure containing records belonging to the dataset identified by `datasetID` belonging to the dataset group identified by `datasetGroupID` and satisfying the filtering criteria expressed by `filter`. The formalism used for the output parameter specifies that an array of genomic records shall be returned, but it does not specify the actual mechanism to return such array of records. The return mechanism can be implemented in different ways according to the specific implementation, for instance all records can be returned together at the same time or they can be returned in multiple chunks received at different times. Both stateful and stateless implementations are supported according to the different type of output mechanism implemented.

Authorization

The action request is tested against the privacy rules defined for the dataset group, and then against the privacy rules defined for the dataset (adding the parameter with the result of the first test as defined in 7.3). If the result of testing the request against the dataset's privacy rules is not permitted, then the call is not authorized. In order to test this action, two new attributes are added to the request: `urn:mpeg:mpeg-g:protection:start` and `urn:mpeg:mpeg-g:protection:end`, of type integer and equal to the values of the fields in the filter. No attribute is added to the request for any other member of the filter structure, except if said so through another channel.

Only data from Access Units for which the access is granted may be returned. In order to know if the access unit is granted, for each access unit the same request as for the dataset is tested against the dataset rules, but changing the attributes: `urn:mpeg:mpeg-g:protection:start` and `urn:mpeg:mpeg-g:protection:end` to be equal to the start and end of the access unit. A new attribute is added, `urn:mpeg:mpeg-g:protection:class`, with type integer and value equal to the access unit type. The access to an unmapped access unit is not granted for this call.

Parameters

| Name | Description |
|-----------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>datasetGroupID</code> | Identifier of the dataset group to be considered for the selective access as specified in ISO/IEC 23092-1. The valid range is $[0, 2^8 - 1]$. Out of range values return an error. |
| <code>datasetID</code> | Identifier of the dataset to be considered for the selective access as specified in ISO/IEC 23092-1. The valid range is $[0, 2^{16} - 1]$. Out of range values return an error. |
| <code>filter</code> | Structure expressing the filtering criteria as specified in subclause 10.3.5 |
| <code>outputRecords</code> | Data structure containing the records matching the filtering criteria. |

Return values

A call to this operation shall return one of these return codes: G_SUCCESS, G_END_OF_DATA, G_NOT_AUTHORIZED, G_VERIFICATION_FAILED, G_DECRYPTION_FAILED, G_DATASETGROUPOUTFOUND, G_DATASET_NOTFOUND, G_REFERENCE_NOTFOUND, G_SEQUENCE_NOTFOUND, G_INVALID_REFERENCE, G_INVALID_BITSTREAM.

10.3.6.4 GetDataByAdvancedFilter

Signature

```
returnCodeT GetDataByAdvancedFilter (
    [in] uintdatasetGroupID,
    [in] uintdatasetID,
    [in] advancedFilterT filter,
    [out] outRecordsT outputRecords
);
```

Description

This function returns an `outRecordsT` structure containing records belonging to the dataset identified by `datasetID` belonging to the dataset group identified by `datasetGroupID` and satisfying the filtering criteria expressed by `filter`. The formalism used for the output parameter specifies that an array of genomic records shall be returned, but it does not specify the actual mechanism to return such array of records. The return mechanism can be implemented in different ways according to the specific implementation, for instance all records can be returned together at the same time or they can be returned in multiple chunks received at different times. Both stateful and stateless implementations are supported according to the different type of output mechanism implemented.

Authorization

The action request is tested against the privacy rules defined for the dataset group, and then against the privacy rules defined for the dataset (adding the parameter with the result of the first test as defined in 7.3). If the result of testing the request against the dataset's privacy rules is not permitted, then the call is not authorized. In order to test this action, two new attributes are added to the request: `urn:mpeg:mpeg-g:protection:start` and `urn:mpeg:mpeg-g:protection:end`, of type integer and equal to the values of the fields in the filter. No attribute is added to the request for any other member of the filter structure, except if said so through another channel.

Only data from Access Units for which the access is granted may be returned. In order to know if the access unit is granted, for each access unit the same request as for the dataset is tested against the rules, but changing the attributes `urn:mpeg:mpeg-g:protection:start` and `urn:mpeg:mpeg-g:protection:end` to be equal to the start and end of the access unit. A new attribute is added, `urn:mpeg:mpeg-g:protection:class`, with type integer and value equal to the access unit type. The access to an unmapped access unit is not granted for this call.

Parameters

| Name | Description |
|-----------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>datasetGroupID</code> | Identifier of the dataset group to be considered for the selective access as specified in ISO/IEC 23092-1. The valid range is $[0, 2^8 - 1]$. Out of range values return an error. |
| <code>datasetID</code> | Identifier of the dataset to be considered for the selective access as specified in ISO/IEC 23092-1. The valid range is $[0, 2^{16} - 1]$. Out of range values return an error. |
| <code>filter</code> | Structure expressing the filtering criteria as specified in subclause 10.3.5 |
| <code>outputRecords</code> | Data structure containing the records matching the filtering criteria. |

Return values

A call to this operation shall return one of these error codes: `G_SUCCESS`, `G_NOT_AUTHORIZED`, `G_VERIFICATION_FAILED`, `G_DECRYPTION_FAILED`, `G_DATASETGROUPTNOTFOUND`, `G_DATASET_NOTFOUND`, `G_REFERENCE_NOTFOUND`, `G_SEQUENCE_NOTFOUND`, `G_INVALID_REFERENCE`, `G_INVALID_BITSTREAM`.

10.3.6.5 GetDataBySignature

Signature

```
returnCodeT GetDataBySignature (
    [in] uintdatasetGroupID,
    [in] uintdatasetID,
    [in] st(v) signature,
    [out] outRecordsT outputRecords
);
```

Description

On successful return, `outputRecords` contains MPEG-G records from the dataset identified by `datasetID` belonging to the dataset group identified by `datasetGroupID` and belonging to access units associated with the string `signature`. The formalism used for the output parameter specifies that an array of genomic records shall be returned, but it does not specify the actual mechanism to return such array of records. The return mechanism can be implemented in different ways according to the specific implementation, for instance all records can be returned together at the same time or they can be returned in multiple chunks received at different times. Both stateful and stateless implementations are supported according to the different type of output mechanism implemented.

Authorization

The action request is tested against the privacy rules defined for the dataset group, and then against the privacy rules defined for the dataset (adding the parameter with the result of the first test as defined in 7.3). If the result of testing the request against the dataset's privacy rules is not permitted, then the call is not authorized. In order to test this action, one new attributes is added to the request: `urn:mpeg:mpeg-g:protection:signature`, of type string and equal to the values of the fields in the filter. No attribute is added to the request for any other member of the filter structure, except if said so through another channel.

Only data from Access Units for which the access is granted may be returned. In order to know if the access unit is granted, for each access unit the same request as for the dataset is tested against the rules, but changing the attribute `urn:mpeg:mpeg-g:protection:signature` to be equal to the one of the signature of the Access Unit. The access is granted if for all signatures the request yields a permit. The access to an aligned access unit is not granted for this call.

Parameters

| Name | Description |
|-----------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>datasetGroupID</code> | Identifier of the dataset group to be considered for the selective access as specified in ISO/IEC 23092-1. The valid range is $[0, 2^8 - 1]$. Out of range values return an error. |
| <code>datasetID</code> | Identifier of the dataset to be considered for the selective access as specified in ISO/IEC 23092-1. The valid range is $[0, 2^{16} - 1]$. Out of range values return an error. |
| <code>signature</code> | Signature identifying access units of Class U as specified in ISO/IEC 23092-1. |
| <code>outputRecords</code> | Data structure containing the records belonging to the access units associated with the specified signature. |

Return values

A call to this operation shall return one of these return codes: `G_SUCCESS`, `G_NOT_AUTHORIZED`, `G_VERIFICATION_FAILED`, `G_DECRYPTION_FAILED`, `G_DATASETGROUPOUTNOTFOUND`, `G_DATASETNOTFOUND`, `G_INVALID_BITSTREAM`.

10.3.6.6 GetDataByLabel

Signature

```
returnCodeT GetDataByLabel (
    [in] uintdatasetGroupID,
    [in] st(v) labelID,
    [out] outRecordsT outputRecords[]
);
```

Description

On successful return, `outputRecords` contains `outRecordsT` structures from the dataset group identified by `datasetGroupID` and belonging to the genomic regions listed in the label identified by `labelID`. The formalism used for the output parameter specifies that an array of genomic records shall be returned, but it does not specify the actual mechanism to return such array of records. The return mechanism can be implemented in different ways according to the specific implementation, for instance all records can be returned together at the same time or they can be returned in multiple chunks received at different times. Both stateful and stateless implementations are supported according to the different type of output mechanism implemented.

Authorization

For each region defined in the label, the authorization of `GetDataBySimpleFilter` is tested, using as start and end the start and end of the region. If for none of the regions the result is permitted, then the right to execute this action is not granted. If for at least one of the regions it was granted, then the right to execute this action is granted. Only those results for which the `GetDataBySimpleFilter` would have been granted can be included in the output.

Parameters

| Name | Description |
|-----------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>datasetGroupID</code> | Identifier of the dataset group the label belongs to as specified in ISO/IEC 23092-1. |
| <code>labelID</code> | Name of the label as specified in ISO/IEC 23092-1. |
| <code>outputRecords</code> | Data structure containing the records belong to the <code>labelID</code> in all datasets of the dataset group identified by <code>datasetGroupID</code> . |

Return values

A call to this operation shall return one of these return codes: `G_SUCCESS`, `G_NOT_AUTHORIZED`, `G_VERIFICATION_FAILED`, `G_DECRYPTION_FAILED`, `G_DATASETGROUPTNOTFOUND`, `G_LABEL_NOTFOUND`, `G_REFERENCE_NOTFOUND`, `G_SEQUENCE_NOTFOUND`, `G_INVALID_REFERENCE`, `G_INVALID_BITSTREAM`.

10.3.7 Metadata

10.3.7.1 GetMetadataFields

Signature

```
returnCodeT GetMetadataFields (
    [in] uintdatasetGroupID,
    [in] uintdatasetID,
    [out] st(v) outputMetadata[]
);
```

Description

On success the array `outputMetadata` contains the names of all the metadata fields set in the dataset identified by `datasetID`.

The function only retrieves those metadata fields names for which the caller is authorized to get the content.

Parameters

| Name | Description |
|----------------|---------------------------------------------------------------------------------------------------------------------------------------------|
| datasetGroupID | Identifier of the dataset group to be considered for the selective access as specified in ISO/IEC 23092-1. The maximum value is $2^8 - 1$. |
| datasetID | Identifier of the dataset to be considered for the selective access as specified in ISO/IEC 23092-1. The maximum value is $2^{16} - 1$. |
| outputMetadata | An array of metadata field names. |

Return values

A call to this operation shall return one of these return codes: G_SUCCESS, G_NOT_AUTHORIZED, G_VERIFICATION_FAILED, G_DECRYPTION_FAILED, G_DATASETGROUPTNOTFOUND, G_DATASET_NOTFOUND, G_INVALID_BITSTREAM.

10.3.7.2 GetMetadataContent

Signature

```
returnCodeT GetMetadataContent (
    [in] uintdatasetGroupID,
    [in] uintdatasetID,
    [in] st(v) fieldName,
    [out] st(v) outputMetadata[]
);
```

Description

This function returns a string equal to the content of the metadata field referenced by `fieldName`, from the XML document stored in the dataset metadata box (as defined in ISO/IEC 23092-1) of the dataset identified by `datasetID` in the dataset group identified by `datasetGroupID`. When `fieldName` is empty, all the metadata is returned. Output metadata is retrieved according to the metadata inheritance rules as specified in [6.4](#).

Parameters

| Name | Description |
|----------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| datasetGroupID | Identifier of the dataset group to be considered for the selective access as specified in ISO/IEC 23092-1. The maximum value is $2^8 - 1$. |
| datasetID | Identifier of the dataset to be considered for the selective access as specified in ISO/IEC 23092-1. The maximum value is $2^8 - 1$. |
| fieldName | XPath reference to the metadata field from which the value is to be retrieved. XPath shall return a single node. The type is defined in subclause 10.3.3 . |
| outputMetadata | A string equal to the content of the metadata field. |

Return values

A call to this operation shall return one of these return codes: G_SUCCESS, G_NOT_AUTHORIZED, G_VERIFICATION_FAILED, G_DECRYPTION_FAILED, G_DATASETGROUPTNOTFOUND, G_METADATA_FIELD_NOTFOUND, G_INVALID_METADATA, G_INVALID_BITSTREAM.

10.3.8 Protection

10.3.8.1 GetDatasetGroupProtection

Signature

```
returnCodeT GetDatasetGroupProtection (
    [in] uint    datasetGroupID,
    [out] st(v)  outputProtection
);
```

Description

This function returns a string equal to the XML document whose coded representation is stored in the dataset group protection box (dgpr as specified in [Table 17](#)) associated with the dataset group identified by datasetGroupID.

Parameters

| Name | Description |
|------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| datasetGroupID | Identifier of the dataset group to be considered for the selective access as specified in ISO/IEC 23092-1. The maximum value is $2^8 - 1$. |
| outputProtection | Null terminated byte array containing the XML document whose coded representation is stored in the dataset group protection box (dgpr) of the dataset group identified by datasetGroupID, encoded in UTF-8. |

Return values

A call to this operation shall return one of these error codes: G_SUCCESS, G_NOT_AUTHORIZED, G_VERIFICATION_FAILED, G_DECRYPTION_FAILED, G_DATASETGROUPOUNFOUNO, G_INVALID_METADATA, G_INVALID_BITSTREAM.

10.3.8.2 GetDatasetProtection

Signature

```
returnCodeT GetDatasetProtection (
    [in] uint    datasetGroupID,
    [in] uint    datasetID,
    [out] st(v)  outputProtection
);
```

Description

This function returns a string equal to the XML document whose coded representation is stored in the dataset protection box (dtp as specified in [Table 17](#)) associated with the dataset identified by datasetID in the dataset group identified by datasetGroupID.

Parameters

| Name | Description |
|------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| datasetGroupID | Identifier of the dataset group associated with the dataset associated with the returned protection metadata. The maximum value is $2^8 - 1$. |
| datasetID | Identifier of the dataset associated with the returned protection metadata. The maximum value is $2^{16} - 1$. |
| outputProtection | Null terminated byte array containing the XML document whose coded representation is stored in the dataset protection box (dtp) of the dataset identified by datasetID, encoded in UTF-8. |

Return values

A call to this operation shall return one of these error codes: G_SUCCESS, G_NOT_AUTHORIZED, G_VERIFICATION_FAILED, G_DECRYPTION_FAILED, G_FILE_NOTFOUND, G_DATASETGROUP_NOTFOUND, G_DATASET_NOTFOUND, G_INVALID_BITSTREAM.

10.3.8.3 GetDatasetRegionProtection

Signature

```
returnCodeT GetDatasetRegionProtection (
    [in] uint    datasetGroup_ID,
    [in] uint    datasetID,
    [in] uint    sequenceID,
    [in] uint    startPos,
    [in] uint    endPos,
    [out] outRegionProtectionT    outputProtection[]
);
```

Description

This function returns an array of `outRegionProtectionT`. Each element of the array defines a region intersecting with the boundaries given as input (delimited by `startPos` and `endPos` on the reference sequence identified by `sequenceID` in the dataset identified by `datasetID` in the dataset group identified by `datasetGroupID`), and a list of all `keyNames` which can be used to decrypt any access unit intersecting with the defined region.

Parameters GetDatasetRegionProtection

| Name | Description |
|-------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>datasetGroupID</code> | Identifier of the dataset group associated with the dataset associated with the returned protection metadata. The maximum value is $2^8 - 1$. |
| <code>datasetID</code> | Identifier of the dataset associated with the returned protection metadata. The maximum value is $2^{16} - 1$. |
| <code>sequenceID</code> | Identifier of the reference sequence that shall be queried. |
| <code>startPos</code> | Start position of the queried region. It shall be greater than or equal to <code>startPos</code> otherwise a G_INVALID_PARAMETER error is returned. |
| <code>endPos</code> | End position of the queried region. It shall be less than or equal to <code>endPos</code> otherwise a G_INVALID_PARAMETER error is returned. |
| <code>outputProtection</code> | An array such that for every encrypted access unit belonging to the range given as input, the list of key allowing its decryption is returned. If no encrypted access unit is present in the range, an empty vector is returned. |

Return values

A call to this operation shall return one of these error codes: G_SUCCESS, G_NOT_AUTHORIZED, G_VERIFICATION_FAILED, G_DECRYPTION_FAILED, G_DATASETGROUP_NOTFOUND, G_DATASET_NOTFOUND, G_SEQUENCE_NOTFOUND, G_INVALID_PARAMETER, G_INVALID_BITSTREAM.

10.3.9 Reference

10.3.9.1 GetDatasetReference

Signature

```
returnCodeT GetDatasetReference (
    [in] uint    datasetGroupID,
    [in] uint    datasetID,
    [in] bool    includeSequences,
```

```
[out] outReferenceT  outputReference
);
```

Description

This function returns an `outReferenceT` structure containing a genomic reference in the form of Raw Reference (as defined in ISO/IEC 23092-2) associated with the dataset identified by `datasetID` and belonging to the dataset group identified by `datasetGroupID`. The `outputReference` is populated as specified in the syntax table below.

Parameters

| Name | Description |
|-------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------|
| <code>datasetGroupID</code> | Identifier of the dataset group containing the dataset associated with the returned genomic reference. The maximum value is $2^8 - 1$. |
| <code>datasetID</code> | Identifier of the dataset associated with the returned genomic reference. The maximum value is $2^{16} - 1$. |
| <code>includeSequences</code> | Flag signalling if the sequences have to be included in the returned output as specified in the syntax table below. |
| <code>outputReference</code> | <code>outReferenceT</code> structure containing the returned genomic reference as specified in subclause 10.3.4.4 . |

| Syntax | Type |
|-------------------------------------------|-------|
| <code>outReferenceT {</code> | |
| seqCount | uint |
| for (seqId=0; seqId<seq_count; seqId++) { | |
| sequenceName | st(v) |
| seqStart | uint |
| seqEnd | uint |
| if(includeSequences){ | |
| for (i=0; i<=seq_end - seq_start; i++) | |
| { | |
| refSequence [seqId][i] | char |
| } | |
| } | |
| } | |
| } | |

Return values

A call to this operation shall return one of these error codes: `G_SUCCESS`, `G_NOT_AUTHORIZED`, `G_VERIFICATION_FAILED`, `G_DECRYPTION_FAILED`, `G_FILE_NOTFOUND`, `G_DATASETGROUPTNOTFOUND`, `G_DATASET_NOTFOUND`, `G_INVALID_BITSTREAM`.

10.3.10 Statistics

10.3.10.1 GetSimpleStatistics

Signature

```
returnCodeT GetSimpleStatistics (
    [in] uint    datasetGroupID,
    [in] uint    datasetID,
```

```

[in] uint    sequenceID,
[in] uint    startPos,
[in] uint    endPos,
[out] uint   maxSegments,
[out] simpleSegmentStatisticsT  outputStatistics[]
);

```

Description

Taking into account the genomic range specified by `startPos` and `endPos` on the reference sequence identified by `sequenceID` in the dataset identified by `datasetID` and belonging to the dataset group identified by `datasetGroupID`, this function returns: the maximum number of segments `maxSegments` for all reads present in the selected records; an array of `maxSegments` structures of type `simpleSegmentStatisticsT` containing statistics associated with the reads present in the selected records. The *i*-th element of output statistics contains statistics for the *i*-th segment of all reads.

Authorization

The action request is tested against the privacy rules defined for the dataset group, and then against the privacy rules defined for the dataset (adding the parameter with the result of the first test as defined in 7.3). If the result of testing the request against the dataset's privacy rules is not permitted, then the call is not authorized.

Only data from Access Units for which the access is granted may be returned. In order to know if the access unit is granted, for each access unit the same request as for the dataset is tested against the rules, but changing the attributes `urn:mpeg:mpeg-g:protection:startPos` and `urn:mpeg:mpeg-g:protection:endPos` to be equal to the start and end of the access unit. The access to an unmapped access unit is not granted for this call.

Parameters

| Name | Description |
|-------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>datasetGroupID</code> | Identifier of the dataset group containing the dataset associated with the returned genomic reference. The maximum value is $2^8 - 1$. |
| <code>datasetID</code> | Identifier of the dataset associated with the returned genomic reference. The maximum value is $2^{16} - 1$. |
| <code>sequenceID</code> | Identifier of the reference sequence that shall be queried. |
| <code>startPos</code> | Start position of the queried region. It shall be less than or equal to <code>endPos</code> otherwise a <code>G_INVALID_PARAMETER</code> error is returned. |
| <code>endPos</code> | End position of the queried region. It shall be greater than or equal to <code>startPos</code> otherwise a <code>G_INVALID_PARAMETER</code> error is returned. |
| <code>maxSegments</code> | Maximum number of segments for all reads present in the selected records |
| <code>outputStatistics</code> | Array of <code>maxSegments</code> <code>simple_segment_statistics_t</code> structures containing the returned statistics as specified in subclause 10.3.4.6. Element 0 in the array contains statistics for the first segment of all reads, and so on. |

Return values

A call to this operation shall return one of these error codes: `G_SUCCESS`, `G_NOT_AUTHORIZED`, `G_VERIFICATION_FAILED`, `G_DECRYPTION_FAILED`, `G_FILE_NOTFOUND`, `G_DATASETGROUPT_NOTFOUND`, `G_DATASET_NOTFOUND`, `G_INVALID_BITSTREAM`.

10.3.10.2 GetAdvancedStatistics

Signature

```

returnCodeT GetAdvancedStatistics (
    [in] uint    datasetGroupID,
    [in] uint    datasetID,
    [in] uint    sequenceID,

```

```

[in] uint    startPos,
[in] uint    endPos,
[out] uint   maxSegments,
[out] advancedSegmentStatisticsT outputStatistics[]
);

```

Description

Taking into account the genomic range specified by `startPos` and `endPos` on the reference sequence identified by `sequenceID` in the dataset identified by `datasetID` and belonging to the dataset group identified by `datasetGroupID`, this function returns: The maximum number of segments `maxSegments` for all reads present in the selected records; an array of `maxSegments` structures of type `advancedSegmentStatisticsT` containing statistics associated with the reads present in the selected records. The *i*-th element of output statistics contains statistics for the *i*-th segment of all reads.

Authorization

The action request is tested against the privacy rules defined for the dataset group, and then against the privacy rules defined for the dataset (adding the parameter with the result of the first test as defined in 7.3). If the result of testing the request against the dataset's privacy rules is not permitted, then the call is not authorized.

Only data from Access Units for which the access is granted may be returned. In order to know if the access unit is granted, for each access unit the same request as for the dataset is tested against the dataset rules, but changing the attributes `urn:mpeg:mpeg-g:protection:startPos` and `urn:mpeg:mpeg-g:protection:endPos` to be equal to the start and end of the access unit. The access to an unmapped access unit is not granted for this call.

Parameters GetStatistics

| Name | Description |
|-------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>datasetGroupID</code> | Identifier of the dataset group containing the dataset associated with the returned genomic reference. The maximum value is $2^8 - 1$. |
| <code>datasetID</code> | Identifier of the dataset associated with the returned genomic reference. The maximum value is $2^{16} - 1$. |
| <code>sequenceID</code> | Identifier of the reference sequence that shall be queried. |
| <code>startPos</code> | Start position of the queried region. It shall be greater than or equal to <code>startPos</code> otherwise a <code>G_INVALID_PARAMETER</code> error is returned. |
| <code>endPos</code> | End position of the queried region. It shall be less than or equal to <code>endPos</code> otherwise a <code>G_INVALID_PARAMETER</code> error is returned. |
| <code>maxSegments</code> | Maximum number of segments for all reads present in the selected records |
| <code>outputStatistics</code> | <code>advancedSegmentStatisticsT</code> structure containing the returned statistics as specified in subclause 0. |

Return values

A call to this operation shall return one of these error codes: `G_SUCCESS`, `G_NOT_AUTHORIZED`, `G_VERIFICATION_FAILED`, `G_DECRYPTION_FAILED`, `G_FILE_NOTFOUND`, `G_DATASETGROUPT_NOTFOUND`, `G_DATASET_NOTFOUND`, `G_INVALID_BITSTREAM`.

Annex A (normative)

XML schemas corresponding to metadata information and protection elements

A.1 Dataset group metadata dgmd XML schema

This schema is available at <https://standards.iso.org/iso-iec/23092/-3/ed-1/en>.

A.2 Dataset metadata dtmd XML schema

This schema is available at <https://standards.iso.org/iso-iec/23092/-3/ed-1/en>.

A.3 Dataset group protection gen_info XML schema

This schema is available at <https://standards.iso.org/iso-iec/23092/-3/ed-1/en>.

A.4 Dataset protection gen_info XML schema

This schema is available at <https://standards.iso.org/iso-iec/23092/-3/ed-1/en>.

A.5 Access unit protection gen_info XML schema

This schema is available at <https://standards.iso.org/iso-iec/23092/-3/ed-1/en>.

A.6 Descriptor stream protection gen_info XML schema

This schema is available at <https://standards.iso.org/iso-iec/23092/-3/ed-1/en>.

A.7 Dataset group reference metadata XML schema

This schema is available at <https://standards.iso.org/iso-iec/23092/-3/ed-1/en>.

Annex B (informative)

XML schemas and XML-based data

B.1 General

This annex describes XML Schemas corresponding to metadata information profiles and extensions. It also includes examples of privacy rules and authorization requests.

B.2 EGA sample extension XML schema

This schema is available at <https://standards.iso.org/iso-iec/23092/-3/ed-1/en>.

B.3 EGA experiment extension XML schema

This schema is available at <https://standards.iso.org/iso-iec/23092/-3/ed-1/en>.

B.4 Genomic data commons extension XML schema

This schema is available at <https://standards.iso.org/iso-iec/23092/-3/ed-1/en>.

B.5 Labels obfuscation XML schema

This schema is available at <https://standards.iso.org/iso-iec/23092/-3/ed-1/en>.

B.6 Sequences obfuscation XML schema

This schema is available at <https://standards.iso.org/iso-iec/23092/-3/ed-1/en>.

B.7 Privacy rules and authorization requests

This annex describes a XACML policy including several privacy rules and several XACML authorization requests.

The privacy policy described provides different rules for giving access to different roles. The rules meaning is summarized next.

The first rule allows the role researcher to perform any action.

The second rule allows the role practitioner to execute the operation GetDataDataset under the following conditions together (intended to protect regions helping to identify Alzheimer's disease predisposition):

- During 2018 (before 2019-01-01).
- No presence of multiple alignments (this attribute should be false).
- Score equal to or better than 0.5.
- Threshold mismatch less than or equal to 5.

- For Class Type equal to 3.
- For a read count of 5000.
- For reference sequence equal to 4, considering range between 40810027 and 41216714, both included, or reference sequence equal to 6, considering range between 26087281 and 26098343, both included.
- Under an emergency situation.

The last rule denies any other combination of role and/or action.

Moreover, several authorization requests are defined. The first one provides an example of valid authorization request for role researcher. The second one provides an example of valid authorization request for role practitioner. The third one provides an example of authorization request for role lab technician, which is not authorized according to the rules defined in the privacy policy.

Policy including different rules for different roles:

```
<?xml version="1.0" encoding="UTF-8"?>
<Policy xmlns="urn:oasis:names:tc:xacml:3.0:core:schema:wd-17"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="urn:oasis:names:tc:xacml:3.0:core:schema:wd-17
  http://docs.oasis-open.org/xacml/3.0/xacml-core-v3-schema-wd-17.xsd"
  PolicyId="urn:isdcm:policyid:1"
  RuleCombiningAlgId="urn:oasis:names:tc:xacml:1.0:rule-combining-algorithm:first-
applicable"
  Version="1.0">
  <Description> Policy getDataDataset </Description>
  <Target/>

  <Rule RuleId="urn:oasis:names:tc:xacml:2.0:ejemplo:RuleMed" Effect="Permit">
    <Description> Any action for researcher is permitted </Description>
    <Target>
      <AnyOf>
        <AllOf>
          <!-- Which kind of user: researcher -->
          <Match MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
```

```

        <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string"
            >researcher</AttributeValue>
        <AttributeDesignator MustBePresent="false"
            Category="urn:oasis:names:tc:xacml:1.0:subject-category:access-subject"

AttributeId="urn:oasis:names:tc:xacml:3.0:example:attribute:role"
            DataType="http://www.w3.org/2001/XMLSchema#string"/>
    </Match>
</AllOf>
</AnyOf>
</Target>
</Rule>

<Rule RuleId="urn:oasis:names:tc:xacml:2.0:ejemplo:RuleGen" Effect="Permit">
    <Description> Get Data from Dataset for practitioner under Emergency situation </
Description>
    <Target>
        <AnyOf>
            <AllOf>

                <!-- Which kind of user: practitioner -->
                <Match MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
                    <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string"
                        >practitioner</AttributeValue>
                    <AttributeDesignator MustBePresent="true"
                        Category="urn:oasis:names:tc:xacml:1.0:subject-category:access-subject"

AttributeId="urn:oasis:names:tc:xacml:3.0:example:attribute:role"
                        DataType="http://www.w3.org/2001/XMLSchema#string"/>
                </Match>

                <!-- Which action -->
                <Match MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
                    <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string"
                        >GetDataDataset</AttributeValue>
                    <AttributeDesignator MustBePresent="true"
                        Category="urn:oasis:names:tc:xacml:3.0:attribute-category:action"
                        AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"
                        DataType="http://www.w3.org/2001/XMLSchema#string"/>
                </Match>
            </AllOf>
        </AnyOf>
    </Target>
</Rule>

```

```

        </Match>
    </AllOf>
</AnyOf>
</Target>
<Condition>
    <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:and">
        <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:date-less-than-
or-equal">
            <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:date-one-
and-only">
                <AttributeDesignator MustBePresent="true"
                    Category="urn:oasis:names:tc:xacml:3.0:date"
AttributeId="accessDate"
                    DataType="http://www.w3.org/2001/XMLSchema#date"/>
            </Apply>
            <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#date"
                >2019-01-01</AttributeValue>
        </Apply>
        <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:boolean-equal">
            <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:boolean-one-
and-only">
                <AttributeDesignator MustBePresent="true" Category="alignment"
                    AttributeId="presence_of_multiple_alignments"
                    DataType="http://www.w3.org/2001/XMLSchema#boolean"/>
            </Apply>
            <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#boolean"
                >false</AttributeValue>
        </Apply>
        <Apply
            FunctionId="urn:oasis:names:tc:xacml:1.0:function:double-greater-
than-or-equal">
            <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:double-one-
and-only">
                <AttributeDesignator MustBePresent="true" Category="score"
                    AttributeId="alignment_score"
                    DataType="http://www.w3.org/2001/XMLSchema#double"/>
            </Apply>
            <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#double"
                >0.5</AttributeValue>
        </Apply>
    </Apply>

```

```

    <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:integer-less-
than-or-equal">
        <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:integer-one-
and-only">
            <AttributeDesignator MustBePresent="true" Category="threshold"
                AttributeId="mismatch_threshold"
                DataType="http://www.w3.org/2001/XMLSchema#integer"/>
            </Apply>
            <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#integer"
                >5</AttributeValue>
        </Apply>
    <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:integer-equal">
        <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:integer-one-
and-only">
            <AttributeDesignator MustBePresent="true" Category="type"
                AttributeId="class_type"
                DataType="http://www.w3.org/2001/XMLSchema#integer"/>
            </Apply>
            <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#integer"
                >3</AttributeValue>
        </Apply>
    <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:integer-less-
than-or-equal">
        <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:integer-one-
and-only">
            <AttributeDesignator MustBePresent="true" Category="count"
                AttributeId="read_count"
                DataType="http://www.w3.org/2001/XMLSchema#integer"/>
            </Apply>
            <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#integer"
                >5000</AttributeValue>
        </Apply>
    <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
        <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-one-
and-only">
            <AttributeDesignator MustBePresent="true"
                Category="urn:oasis:names:tc:xacml:3.0:string"
                AttributeId="situation"
                DataType="http://www.w3.org/2001/XMLSchema#string"/>
            </Apply>
            <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string"
                >Emergency</AttributeValue>
        </Apply>
    </Apply>

```

```

        <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:or">
            <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:and">
                <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:integ
er-equal">
                    <Apply
                        FunctionId="urn:oasis:names:tc:xacml:1.0:function:integ
er-one-and-only">
                            <AttributeDesignator MustBePresent="true"
                                Category="sequence"
                                    AttributeId="reference_sequence"
                                        DataType="http://www.w3.org/2001/XMLSchema#integer"/>
                                </Apply>
                                    <AttributeValue
                                        DataType="http://www.w3.org/2001/XMLSchema#integer"
                                            >4</AttributeValue>
                                </Apply>
                                    <Apply
                                        FunctionId="urn:oasis:names:tc:xacml:1.0:function:integer-
greater-than-or-equal">
                                            <Apply
                                                FunctionId="urn:oasis:names:tc:xacml:1.0:function:integ
er-one-and-only">
                                                    <AttributeDesignator MustBePresent="true"
                                                        Category="urn:oasis:names:tc:xacml:3.0:integer"
                                                            AttributeId="position"
                                                                DataType="http://www.w3.org/2001/XMLSchema#integer"/>
                                                        </Apply>
                                                            <AttributeValue DataType="http://www.w3.org/2001/
XMLSchema#integer"
                                                                >40810027</AttributeValue>
                                                        </Apply>
                                                            <Apply
                                                                FunctionId="urn:oasis:names:tc:xacml:1.0:function:integer-
less-than-or-equal">
                                                                    <Apply
                                                                        FunctionId="urn:oasis:names:tc:xacml:1.0:function:integ
er-one-and-only">
                                                                            <AttributeDesignator MustBePresent="true"

```

```

        Category="urn:oasis:names:tc:xacml:3.0:integer"
        AttributeId="position"
        DataType="http://www.w3.org/2001/XMLSchema#integer"/>
    </Apply>
    <AttributeValue DataType="http://www.w3.org/2001/
XMLSchema#integer"
        >41216714</AttributeValue>
    </Apply>
</Apply>
<Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:and">
    <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:integ
er-equal">
        <Apply
            FunctionId="urn:oasis:names:tc:xacml:1.0:function:integ
er-one-and-only">
                <AttributeDesignator MustBePresent="true"
                    Category="sequence"
                    AttributeId="reference_sequence"
                    DataType="http://www.w3.org/2001/XMLSchema#integer"/>
            </Apply>
            <AttributeValue DataType="http://www.w3.org/2001/
XMLSchema#integer"
                >6</AttributeValue>
            </Apply>
            <Apply
                FunctionId="urn:oasis:names:tc:xacml:1.0:function:integer-
greater-than-or-equal">
                    <Apply
                        FunctionId="urn:oasis:names:tc:xacml:1.0:function:integer-one-and-only">
                            <AttributeDesignator MustBePresent="true"
                                Category="urn:oasis:names:tc:xacml:3.0:integer"
                                AttributeId="position"
                                DataType="http://www.w3.org/2001/XMLSchema#integer"/>
                            </Apply>
                            <AttributeValue DataType="http://www.w3.org/2001/
XMLSchema#integer"
                                >26087281</AttributeValue>
                            </Apply>
                            <Apply
                                FunctionId="urn:oasis:names:tc:xacml:1.0:function:integer-
less-than-or-equal">
                                    <Apply

```