International Standard

ISO 24138

First edition
2024-05

Information and documentation — International Standard Content Code (ISCC)

Information et documentation — Code international normalisé de contenu (ISCC)

Reference number
ISO 24138:2024(en)

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

Published in Switzerland

Contents

Page

Foreword

ISO (the International Organization for Standardization) is a worldwide federation of national standards bodies (ISO member bodies). The work of preparing International Standards is normally carried out through ISO technical committees. Each member body interested in a subject for which a technical committee has been established has the right to be represented on that committee. International organizations, governmental and non-governmental, in liaison with ISO, also take part in the work. ISO collaborates closely with the International Electrotechnical Commission (IEC) on all matters of electrotechnical standardization.

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 ISO 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).

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

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

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

This document was prepared by Technical Committee ISO/TC 46, *Information and documentation*, Subcommittee SC 9, *Identification and description.*

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

While ISO/TC 46/SC 9 has established a variety of specific identifier standards, a content-dependent identifier for digital assets in all content formats has not yet been agreed.

Digital content is dynamic, always in motion, and acted upon globally by a variety of entities with different interests and requirements. Digital content continuously re-encodes, resizes, and re-compresses, changing its data as it travels through a complex network of actors and systems.

The International Standard Content Code (ISCC) is an identifier for numerous types of digital assets. An ISCC-CODE is generated from the digital content itself. It is the result of processing the digital content using a variety of algorithms including hash algorithms. The generated ISCC-CODE supports data integrity verification and preserves an estimate of the data, digital content and metadata similarity. However, ISCC has different functionality from content recognition systems.

The ISCC supports the association of higher-level identifiers (like work and product identifiers) with the digitally encoded manifestations of content. The ISCC does not specify a system for managing authoritative metadata. Other content identifier standards can use ISCC to support discoverability of their identifiers and metadata based on digital content.

Organizations, individuals and machines may generate ISCCs for numerous kinds of digital assets and use them for identification and management of those assets.

ISCCs are neither manually nor automatically assigned to digital media assets. Instead, ISCCs are derived from media assets according to the procedures described in this document. Unrelated parties can independently derive the same ISCC from a given media asset.

ISCCs exclusively reference media assets without any implication about ownership. As such, ISCCs are not managed authoritatively by any institution or entity.

The ISCC enables interoperability between different actors and systems using digital assets and supports scenarios that require content deduplication, database synchronization and indexing, integrity verification, timestamping, versioning, data provenance, similarity clustering, anomaly detection, usage tracking, allocation of royalties, fact-checking and general digital asset management use-cases.

This document includes sections targeting a general audience but also descriptions of more technical procedures.

Future editions of this document can be developed as outlined in Annex C.

Information and documentation — International Standard Content Code (ISCC)

1 Scope

This document specifies the syntax and structure of the International Standard Content Code (ISCC), as an identification system for digital assets (including encodings of text, images, audio, video or other content across all media sectors). It also describes ISCC metadata and the use of ISCC in conjunction with other schemes, such as DOI, ISAN, ISBN, ISRC, ISSN and ISWC.

An ISCC applies to a specific digital asset and is a data-descriptor deterministically constructed from multiple hash digests using the algorithms and rules in this document. This document does not provide information on registration of ISCCs.

2 Normative references

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

ISO/IEC 10646:2020, *Information technology — Universal coded character set (UCS)*

ISO/IEC 15938, *Information technology — Multimedia content description interface*

ISO/IEC 21778, *Information technology — The JSON data interchange syntax*

IETF RFC 4648, *The Base16, Base32, and Base64 Data Encodings*[1)]

IETF RFC 2397, *The "data" URL scheme*[2)]

IETF RFC 8785, *JSON Canonicalization Scheme (JCS)*[3)]

W3C, C14N 1.1, *Canonical XML Version 1.1*[4)]

W3C, JSON-LD 1.1, *A JSON-based Serialization for Linked Data*[5)]

3 Terms and definitions

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

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

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

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

1) Online available: https://datatracker.ietf.org/doc/html/rfc4648

2) Online available: https://datatracker.ietf.org/doc/html/rfc2397

3) Online available: https://datatracker.ietf.org/doc/html/rfc8785

4) Online available: https://www.w3.org/TR/xml-c14n11

5) Online available: https://www.w3.org/TR/json-ld/

3.1
bit
atomic unit of information in a computer system

3.2
byte
sequence of 8 *bits* (3.1)

3.3
nibble
half a *byte* (3.2), which can be represented by a single hexadecimal digit

[SOURCE: ISO 20038:2017, 3.12]

3.4
data
ordered sequence of *bits* (3.1)

3.5
file
stored *data* (3.4) with a known number of *bits* (3.1) and a filename

3.6
stream
data (3.4) in transit with a known or unknown number of *bits* (3.1)

3.7
content
information organized to provide value to a user

3.8
digital content
manifestation of *content* (3.7) in form of *data* (3.4) structured according to a set of rules

3.9
metadata
data (3.4) that defines and describes other data

[SOURCE: ISO 24531:2013, 4.32]

3.10
seed metadata
initial *metadata* (3.9) used as input to a *hash algorithm* (3.1) function

3.11
content format
set of rules used to structure *digital content* (3.8)

3.12
media type
two-part *identifier* (3.15) specifying the nature of the referenced *data* (3.4)

[SOURCE: ISO/IEC 19757-4:2006, 3.9]

3.13
digital asset
file (3.5) or *stream* (3.6) encoded in conformance with a specific *content format* (3.11)

3.14
referent
object which is identified

3.15
identifier
sequence of characters that uniquely denotes a *referent* (3.14)

3.16
identifier system
system to enable the provision of *identifiers* (3.15) for a given category of *referents* (3.14)

3.17
content identifier
identifier (3.15) whose *referent* (3.14) is *content* (3.7)

3.18
content-dependent identifier
content identifier (3.18) whose *data* (3.4) depends on the *digital content* (3.8) that it identifies

3.19
content recognition system
system whose primary purpose is to recognise *digital content* (3.8) on a granular level

3.20
algorithm
set of instructions

3.21
hash algorithm
deterministic *algorithm* (3.20) that produces fixed-length *data* (3.4) from an input of arbitrary-length data

3.22
hash digest
result of processing *data* (3.4) with a *hash algorithm* (3.21)

3.23
cryptographic hash function
computationally efficient function mapping binary strings of arbitrary length to binary strings of fixed length, such that it is computationally infeasible to find two distinct values that hash into a common value

3.24
similarity hash
hash digest (3.22) that preserves correlations between inputs to the *hash algorithm* (3.21)

3.25
content defined chunking
CDC
method to split *data* (3.4) into variable length chunks based on internal features such that chunk boundaries are more resistant to *byte* (3.2) shifting

3.26
actor
human or non-human (hardware or software) entity that interacts with a system

3.27
Merkle tree
tree data structure in which every leaf node is labelled with the *hash digest* (3.22) of a data element and every non-leaf node is labelled with the hash digest of the labels of its child nodes

3.28
Merkle root
root node of a *Merkle tree* (3.27)

[SOURCE: ISO 22739:2024, 3.57]

3.29
ISCC processor
application that generates ISCCs for *digital content* (3.8)

3.30
plain text
data (3.4) with a known text encoding that can be transcoded to Unicode

3.31
whitespace
nondisplaying formatting characters such as spaces, tabs, etc., that are embedded within a block of free text

[SOURCE: ISO/IEC/IEEE 31320-2:2012, 3.1.210]

4 Structure and format of the ISCC

4.1 General structure

a) An ISCC shall be composed of an ISCC-HEADER and an ISCC-BODY (see Figure 1).

b) The ISCC-HEADER shall describe the MainType, SubType, Version, and Length of its ISCC-BODY.

c) An ISCC-UNIT shall be an ISCC based on one specific algorithm.

d) An ISCC-CODE shall be an ISCC composed from two or more different ISCC-UNITs.

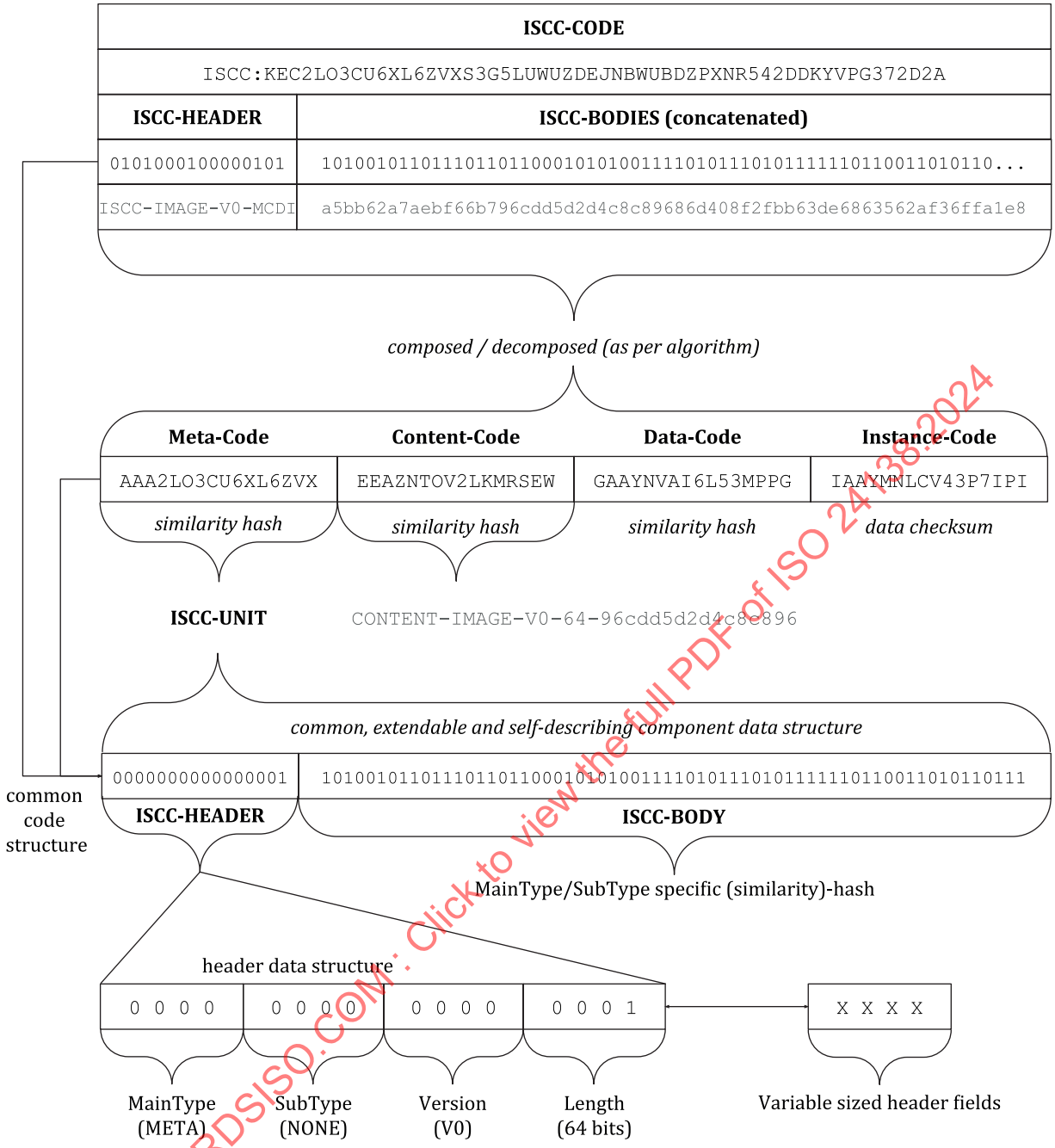

Figure 1 — General structure of an ISCC

The concatenation of the ISCC-UNITs is based on the underlying data and is not visible in the string representation of the ISCC-CODE itself. See 6.6 on how an ISCC-CODE is composed from individual ISCC-UNITs.

4.2 ISCC-HEADER

4.2.1 General

4.2.1.1 The ISCC-HEADER is a variable sized bitstream composed of an ordered sequence of the 4 header-fields MainType, SubType, Version, Length.

4.2.1.2 Each header-field is a bitstream with a length between 4 and 16 bits and encodes an integer value between 0 and 4679 (see Table 1) with the following encoding scheme.

a) The total bit-length of a header-field shall be determined by its prefix-bits.

b) The prefix-bits shall be followed by data-bits.

c) The data-bits shall be interpreted as unsigned integer values plus the maximum value of the preceding range.

d) If the total length of all header fields in number of bits is not divisible by 8, the header shall be padded with 4 zero bits (0000) on the right side.

Table 1 — Variable length ISCC-HEADER field encoding

Prefix bits	Number of nibbles	Number of data bits	Integer range
0	1	3	0-7
10	2	6	8-71
110	3	9	72-583
1110	4	12	584-4679

4.2.1.3 The interpretation of the integer value of a header-field shall be context dependent.

a) For the MainType and SubType fields, it shall be an identifier for the designated type (see 4.2.2 and 4.2.3).

b) For the Version field it shall be the literal version number (see 4.2.4).

c) For the Length field of ISCC-UNITs, it shall be a number used as a multiplier to calculate the bit length of the ISCC-BODY (see 4.2.5, Table 6).

d) For the Length field of ISCC-CODEs, it shall be a bit-pattern encoding the combination of ISCC-UNITs and the bit-length of the ISCC-BODY (see 4.2.5, Table 7).

EXAMPLE Header Field Examples

0 = 0000

1 = 0001

...

7 = 0111

8 = 1000 0000

9 = 1000 0001

...

4.2.2 MainTypes

The MainType header-field shall signify the type of an ISCC (see Table 2).

Backward incompatible updates to an algorithm associated with a MainType shall be indicated by incrementing the version field of the ISCC-HEADER of the respective MainType.

NOTE This document specifies initial algorithms (version 0) for all reserved MainTypes with the exception of the SEMANTIC type which is not currently defined.

Table 2 — Reserved MainTypes

ID	Symbol	Bits	Definition
0	META	0000	An ISCC-UNIT that matches on metadata similarity
1	SEMANTIC	0001	An ISCC-UNIT that matches on semantic content similarity
2	CONTENT	0010	An ISCC-UNIT that matches on perceptual content similarity
3	DATA	0011	An ISCC-UNIT that matches on data similarity
4	INSTANCE	0100	An ISCC-UNIT that matches on data identity
5	ISCC	0101	An ISCC-CODE composed of two or more headerless ISCC-UNITs for multi-modal matching

4.2.3 SubTypes

The MainTypes META, DATA, and INSTANCE shall have a single default SubType NONE encoded with the bits 0000.

The MainTypes SEMANTIC, CONTENT, and ISCC shall have SubTypes that signify the perceptual mode (see Table 3 and Table 4).

Table 3 — Reserved SubTypes for MainTypes ISCC, SEMANTIC and CONTENT

ID	Symbol	Bits	Definition
0	TEXT	0000	Match on text similarity
1	IMAGE	0001	Match on image similarity
2	AUDIO	0010	Match on audio similarity
3	VIDEO	0011	Match on video similarity
4	MIXED	0100	Match on multi-modal similarity

Table 4 — Additional Reserved SubTypes for the MainType ISCC

ID	Symbol	Bits	Definition
5	SUM	0101	Composite of ISCC-UNITs including only Data- and Instance-Code
6	NONE	0110	Composite ISCC-UNITs including Meta-, Data- and Instance-Code

4.2.4 Version

All ISCC-HEADERs shall have a version header-field of 0000 for the first edition of this document (see Table 5).

Table 5 — Reserved ISCC Versions

ID	Symbol	Bits	Definition
0	V0	0000	Initial version of ISCC-UNITs and ISCC-CODE

4.2.5 Length

4.2.5.1 General

The encoding of the Length header-field shall be specific to the MainType.

4.2.5.2 Length of ISCC-UNITs

For ISCC-UNITs of the MainTypes META, SEMANTIC, CONTENT, DATA and INSTANCE the length value shall be encoded as the number of 32-bit blocks of the ISCC-BODY in addition to the minimum length of 32 bits (see Table 6).

Table 6 — Reserved length field values (multiples of 32 bit)

ID	Symbol	Bits	Definition
0	L32	0000	Length of body is 32 bits (minimum length)
1	L64	0001	Length of body is 64 bits (default length)
2	L96	0010	Length of body is 96 bits
3	L128	0011	Length of body is 128 bits
4	L160	0100	Length of body is 160 bits
5	L192	0101	Length of body is 192 bits
6	L224	0110	Length of body is 224 bits
7	L256	0111	Length of body is 256 bits

4.2.5.3 Length of ISCC-CODEs

a) For ISCC-CODEs, the length value shall designate the composition of ISCC-UNITs (see Table 7).

b) The Data-Code and Instance-Code shall be mandatory 64-bit components of an ISCC-CODE.

c) The first data-bit shall designate the presence of a 64-bit Meta-Code.

d) The second data-bit shall designate the presence of a 64-bit Semantic-Code.

e) The third data-bit shall designate the presence of a 64-bit Content-Code.

f) The length of an ISCC-CODE shall be calculated as the number of active data-bits times 64 plus 128 bits of mandatory data.

Table 7 — Reserved length field values (for MainType ISCC)

ID	Symbol	Bits	Definition
0	SUM	0000	No optional ISCC-UNITs. Length of body is 128 bits.
1	CDI	0001	Includes Content-Code. Length of body is 192 bits
2	SDI	0010	Includes Semantic-Code. Length of body is 192 bits
3	SCDI	0011	Includes Semantic- and Content-Code. Length of body is 256 bits
4	MDI	0100	Includes Meta-Code. Length of body is 192 bits
5	MCDI	0101	Includes Meta-Code and Content-Code. Length of body is 256 bits
6	MSDI	0110	Includes Meta-Code and Semantic-Code. Length of body is 256 bits
7	MSCDI	0111	Includes Meta-, Semantic-, and Content-Code. Length is 320 bits

4.3 ISCC-BODY

a) The preceding MainType, SubType, and Version fields shall qualify the semantics of an ISCC-BODY.

b) The Length field shall determine the number of bits of an ISCC-BODY.

4.4 Encoding

4.4.1 Canonical form

The printable canonical form of an ISCC shall be its RFC 4648 Base32 encoded representation without padding and prefixed with “ISCC:”. Base32 defines an upper case standard alphabet.

EXAMPLE `ISCC:KEC43HJLPUSHVAZT66YLPUWNVACWYPIV533TRQMWF2IUQYSP5LA4CTY`

4.4.2 URI encoding

An ISCC may be encoded using the syntax of a Uniform Resource Identifier (URI) as defined in RFC 3986.

a) The URI representation shall have the format <scheme>:<path>.

b) The URI scheme shall be the string “iscc”.

c) The URI path shall be the lower-cased base32 representation of an ISCC without padding.

EXAMPLE iscc:kec43hjlpushvazt66ylpuwnvacwypiv533trqmwf2iuqysp5la4cty

NOTE Because Base32 defines an upper case standard alphabet, the canonical form differs from the URI form, which is represented in lower case.

4.4.3 Multiformats encoding

The ISCC may be encoded as a multibase[13] string (see Table 8).

a) The multicodec identifier of an ISCC shall be “0xcc01” (see Table 9).

b) A Multiformat representation of an ISCC shall be prefixed with a multibase code.

c) The encoding scheme shall be <multibase><multicodec><iscc-header><iscc-body>.

ISCC shall support the multibase encodings given in Tables 8 and 9.

Table 8 — Supported multibase encodings

Encoding	Code	Definition
base16	f	hexadecimal
base32	b	RFC4648 case-insensitive - no padding
base32hex	v	RFC4648 case-insensitive - no padding - highest char
base58btc	z	base58 bitcoin
base64url	u	RFC4648 no padding

Table 9 — Examples of ISCCs in multiformats encoding

Encoding	Example
MF base16	`fcc01510cd9d2b7d247a8333f7b0b7d2cda8056c3d15eef738c1962e9148624feac1c14f`
MF base32	`bzqavcbontuvx2jd2qmz7pmfx2lg2qblmhuk655zyyglc5ekimjh6vqobj4`
MF base32hex	`vpg0l21edjklnq93qgcpvfc5nqb6qg1bc7kauttpoo6b2t4a8c97ulge19s`
MF base58btc	`z2Yr3BMx3Rj56fyYkNvfa19PCk4SjspQhpVWoLSGg9yXr4vUGsx`
MF base64url	`uzAFRBc2dK30keoMz97C30s2oBWw9Fe73OMGWLpFIYk_qwcFP`

4.4.4 Readable encoding

The ISCC may be encoded in human readable representation.

a) The readable representation shall encode the header fields with their symbols and the ISCC-BODY in base16 lower-case.

b) The header fields and the ISCC-BODY shall be separated with hyphens.

EXAMPLE

ISCC-IMAGE-V0-MCDI-cd9d2b7d247a8333f7b0b7d2cda8056c3d15eef738c1962e9148624feac1c14f

5 ISCC-UNITs

5.1 Meta-Code

5.1.1 General

The Meta-Code is a similarity hash generated from referent seed metadata in accordance with Annex B.

5.1.2 Purpose

The Meta-Code shall support the following use cases:

a) clustering of digital assets based on their metadata;

b) discovery of digital assets with similar metadata;

c) verification or manual disambiguation of matching codes.

5.1.3 Format

The Meta-Code shall have the data format as illustrated in Figure 2:

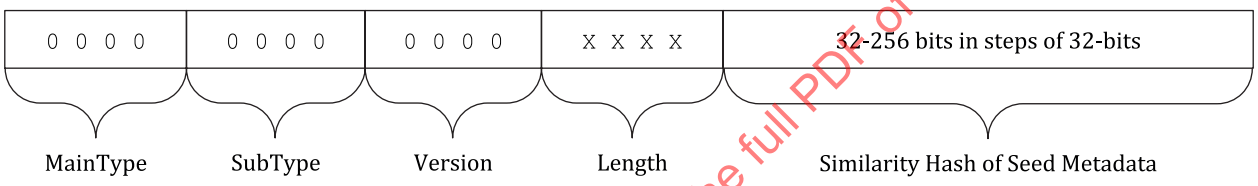

Figure 2 — Data format of the Meta-Code

EXAMPLE 1 64-bit Meta-Code in its canonical form:

ISCC:AAAUL6P7RMVNT4UJ

EXAMPLE 2 256-bit Meta-Code in its canonical form:

ISCC:AADUL6P7RMVNT4UJJ4SMTDXBL5JFZ5XPCDKO42XYPJEVQ4L7PTYDORQ

5.1.4 Inputs

5.1.4.1 General

Seed metadata is the metadata that is used as the input to calculate the Meta-Code and has three possible elements:

a) name (required): the name or title of the work manifested by the digital asset;

b) description (optional): a disambiguating textual description of the digital asset;

c) meta (optional): subject, industry, or use-case specific metadata.

Seed metadata shall be stored and carried along unaltered with ISCC Metadata if automated verification of the Meta-Code based on the original seed metadata is required.

NOTE 1 Because seed metadata is used to construct the Meta-Code, changes to its value can produce different (and therefore no longer matching) Meta-Codes.

NOTE 2 The identifier standards and their schemas, such as DOI, ISAN, ISBN, ISRC, ISSN and ISWC, provide helpful guidance in selecting seed metadata.

5.1.4.2 name element

The text input for the name element shall be pre-processed before similarity hashing as follows.

a) Apply ISO/IEC 10646 NFKC Unicode Normalization (see Unicode Normalization Forms https://unicode.org/reports/tr15/#Norm_Forms).

b) Remove control characters (see Unicode Character Database https://www.unicode.org/ucd/).

c) Strip leading and trailing whitespace.

d) Trim the end of the text such that the UTF-8 encoded size does not exceed 128 bytes.

5.1.4.3 description element

Text input for the description element shall be pre-processed before similarity hashing as follows.

a) Apply NFKC Unicode Normalization.

b) Remove control characters (as specified by Unicode Character Database) except for the following newline characters:

   1) U000A - Line Feed;

   2) U000B - Vertical Tab;

   3) U000C - Form Feed;

   4) U000D - Carriage Return;

   5) U0085 - Next Line;

   6) U2028 - Line Separator;

   7) U2029 - Paragraph Separator.

c) Collapse more than two consecutive newlines characters to a maximum of two consecutive newlines.

d) Strip leading and trailing whitespace characters.

5.1.4.4 meta element

a) The value of the meta element shall be wrapped in an RFC 2397 Data-URL.

b) The value of the meta element may include any conceivable and supportive metadata such as for example:

   1) JSON serialized metadata (data:application/json;base64,<data>);

   2) JSON-LD serialized metadata (data:application/ld+json;base64,<data>);

   3) XML serialized metadata (data:application/xml;base64,<data>);

   4) MARCXML (data:application/marcxml+xml;base64,<data>);

   5) IPTC NewsML (data:application/vnd.iptc.g2.newsitem+xml;base64,<data>);

   6) a file header (data:application/octet-stream;base64,<data>);

   7) a thumbnail image (<data>);

   8) an audio sample (data:audio/mp4;base64,<data>).

c) If the value of the meta element is JSON or JSON-LD, it shall be serialized with RFC 8785 JCS canonicalization before being wrapped in a Data-URL.

d) If the value of the meta element is XML, it shall be serialized as Canonical XML.

e) The Data-URL shall be pre-processed before similarity hashing as follows:

   1) Decode the base64 encoded data section of the Data URL to a raw bitstream without further interpretation.

5.1.5 Outputs

Meta-Code processing shall generate the following output elements for inclusion into the produced ISCC metadata:

a) iscc (required): the ISCC Meta-Code in its canonical form;

b) name (required): the pre-processed value of the name element;

c) meta (optional): the unaltered value of the meta element;

d) description (optional): the pre-processed value of the description element;

e) metahash (required): a cryptographic hash of the seed metadata.

NOTE 1 The reference implementation uses a multihash[12] encoded BLAKE3[10] value for the metahash element.

NOTE 2 An ISCC processor can produce other custom output elements, which are helpful to identify the digital asset.

5.1.6 Seed metadata processing

5.1.6.1 Meta-Code processing

The Meta-Code shall be constructed from 2 similarity hashes interleaved in 32-bit chunks by selecting the elements according to the algorithm illustrated in Figure 3.

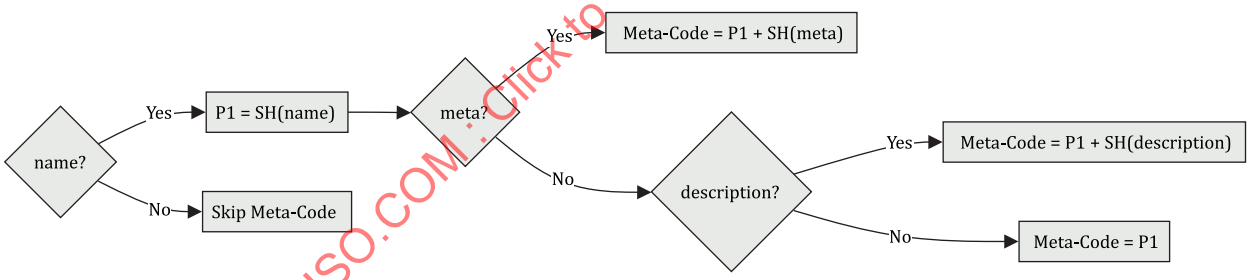

Figure 3 — Meta-Code processing logic

a) If the name element is unavailable, Meta-Code generation shall be skipped.

b) The first part of the similarity hash for the Meta-Code shall be generated from the name element.

c) The second part of the similarity hash shall be generated from the meta element.

d) If the meta element is unavailable, the second part of the similarity hash shall be generated from the description element.

e) If the description element is unavailable, the second part of the similarity hash shall also be generated from the name element.

5.1.6.2 Meta-Hash processing

The Meta-Hash shall be constructed from the seed metadata by selecting input elements according to the algorithm illustrated in Figure 4.

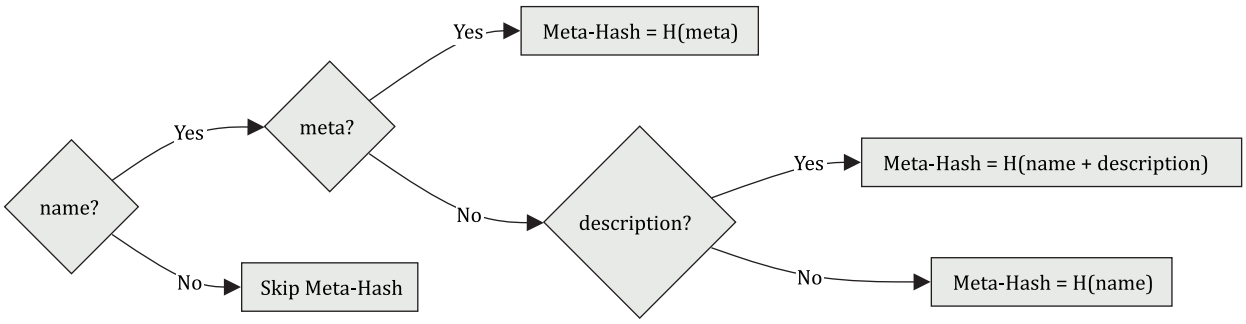

Figure 4 — Meta-Hash processing logic

a) If the name element is unavailable, Meta-Hash generation shall be skipped.

b) If the meta element is available, the decoded raw and un-interpreted data of the Data-URL shall be used as sole input to the cryptographic hash function.

c) If the meta element is unavailable, but the description element is available, the space-concatenated value of the pre-processed name and description shall be the input to the cryptographic hash function.

d) If only the name element is available, its pre-processed value shall be the input to the cryptographic hash function.

5.1.7 Metadata embedding

a) Seed metadata shall be embedded into the processed digital asset if:

   1) seed metadata values have been provided explicitly to an ISCC processor;

   2) the ISCC processor supports metadata embedding for the given media type.

b) If the media type supports ISO 16684 XMP metadata-embedding, an ISCC processor shall use the namespace https://purl.org/iscc/schema and embed seed metadata values under the names:

   1) Xmp.iscc.name

   2) Xmp.iscc.description

   3) Xmp.iscc.meta

c) If the media type does not support ISO 16684 XMP metadata-embedding the ISCC processor may choose other suitable format-specific fields for embedding seed metadata.

d) If seed metadata is to be embedded, it shall be embedded before processing other ISCC-UNITs.

e) An ISCC processor should document for which media types it supports metadata-embedding and how it maps seed metadata to format specific elements.

5.1.8 Metadata extraction

a) An ISCC processor shall try to extract seed metadata from the digital asset if:

   1) seed metadata has not been provided explicitly to the ISCC processor;

   2) the ISCC processor supports metadata extraction for the given media type.

b) Seed metadata shall be extracted with the following precedence:

   1) extract seed metadata from XMP metadata under the namespace https://purl.org/iscc/schema;

   2) extract seed metadata from suitable, format-specific embedded metadata;

3) use the filename of the asset as a value for the name element, discarding the file extension and replacing the characters "-" and "_" with spaces;

4) an ISCC processor shall document for which media types it supports metadata-extraction and how it maps seed metadata to format specific elements.

5.2 Content-Codes

5.2.1 General

a) A Content-Code shall be a similarity hash generated from the referent depending on the content category of the referent.

b) A Content-Code shall be identical or similar for structurally identical or similar perceptible content encoded in different file formats.

c) A Content-Code shall have one of the following SubTypes: text, image, audio, video, mixed as defined in 4.2.3.

d) An ISCC processor shall expose a separate function for each supported SubTypes.

NOTE The high-level process for generating Content-Codes is first to determine the content format and then use the specified media type content extraction, normalization, segmentation and fingerprinting algorithm. The resulting granular fingerprint is the input for generating the final Content-Code.

5.2.2 Purpose

Content-Codes shall support the following use cases:

a) discovery and matching of duplicate content encoded in different formats;

b) clustering near-duplicate digital assets based on structural or perceptual similarity even after compression or transcoding into various content formats.

5.3 Content-Code Subtype Text

5.3.1 General

a) The Content-Code Subtype Text (Text-Code) shall be a Content-Code generated from the plain text content extracted from a digital asset that contains text.

b) The Text-Code shall be robust against text document format conversion and minor edits.

5.3.2 Format

The Text-Code shall have the data format illustrated in Figure 5.

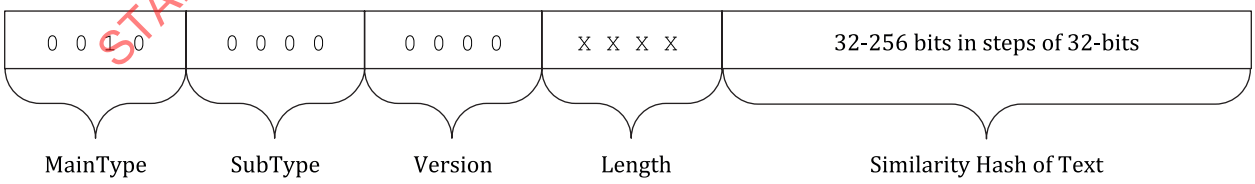

Figure 5 — Data format of the Text-Code

EXAMPLE 1 64-bit Text-Code in its canonical form with text input "Hello World":

ISCC:EAASKDNZNYGUUF5A

EXAMPLE 2 256-bit Text-Code in its canonical form with text input "Hello World":

ISCC:EADSKDNZNYGUUF5AMFEJLZ5P66CP5YKCOA3X7F36RWE4CIRCBTUWXYY

5.3.3 Inputs

a) The input for calculating the Text-Code shall be the plain text as extracted from a digital document.

b) An ISCC implementation may use any text extraction mechanism, including optical character recognition, to extract plain text from a digital document.

c) Plain text used as input for Text-Code calculation shall not include any processing instructions (SGML, HTML, Markdown, and other markup information).

5.3.4 Outputs

Text-Code processing shall generate the following ISCC metadata output elements:

a) iscc: the Text-Code in its canonical form (required);

b) characters: The number of characters of the source text after pre-processing (optional);

c) Additional metadata extracted from the document (optional).

5.3.5 Processing

5.3.5.1 An ISCC processor shall pre-process Text input as follows:

a) Apply NFD Unicode Normalization.

b) Remove all whitespace characters from the text.

c) Convert text to lower case in accordance with Unicode 'Case Folding Properties' (see https://www.unicode.org/Public/UCD/latest/ucd/CaseFolding.txt).

d) Remove all characters from Unicode categories Mark (M), Punctuation (P) and Other (C).

e) Apply NFKC Unicode Normalization.

5.3.5.2 An ISCC processor shall calculate the Text-Code as follows:

a) Split the pre-processed text into n-grams of 13 characters by sliding over the text character-wise.

b) Create a list of 32-bit unsigned integers by hashing the UTF-8 encoded representation of the n-grams using the XXH32 algorithm.

c) Apply the Minhash256 algorithm to the list of integers to calculate the ISCC-BODY of the Text-Code.

5.3.6 Conformance

The normative behaviour of an ISCC processor in generating a Text-Code is specified only for UTF-8 encoded text input. An implementation of the Text-Code algorithm shall be regarded as conforming to the standard as long as it creates the same Text-Code as the reference implementation in accordance to Annex D for the same UTF-8 encoded text input.

The normative behaviour of an ISCC processor in generating a Text-Code from other sources is not specified and different codes can be generated depending on the tools used for text extraction from the source. Implementers seeking to guarantee interoperability with each other in these circumstances should select the same tool for text extraction.

NOTE For further technical details, see source-code in the modules "code_content_text.py" and "minhash.py" of the reference implementation in: https://standards.iso.org/iso/24138/ed-1/en/.

5.4 Content-Code Subtype Image

5.4.1 General

a) The Content-Code Subtype Image (Image-Code) shall be a perceptual similarity hash of the input image.

b) The Image-Code shall be robust against image format conversion, scaling, compression and minor edits.

5.4.2 Format

The Image-Code shall have the data format illustrated in Figure 6.

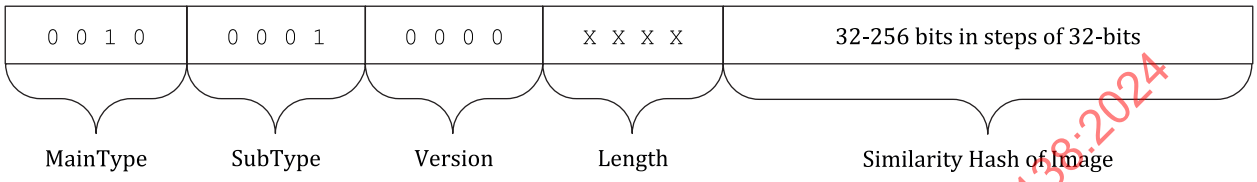

Figure 6 — Data format of the Image-Code

EXAMPLE 1 64-bit Image-Code in its canonical form:

ISCC:EEA4GQZQTY6J5DTH

EXAMPLE 2 256-bit Image-Code in its canonical form:

ISCC:EED4GQZQTY6J5DTHQ2DWCPDZHQOM6QZQTY6J5DTFZ2DWCPDZHQOMXDI

5.4.3 Inputs

a) The input for calculating the Image-Code shall be an image file.

b) An ISCC processor shall at least support the JPEG and PNG image formats.

5.4.4 Outputs

Image-Code processing shall generate the following ISCC metadata output elements:

a) iscc: the Image-Code in its canonical form (required);

b) width: width of the original input image in number of pixels (optional);

c) height: height of the original input image in number of pixels (optional);

d) thumbnail: a thumbnail of the original image encoded as Data-URL (optional);

e) Additional metadata extracted from the image (optional).

5.4.5 Processing

5.4.5.1 An ISCC processor shall pre-process the image file input as follows:

a) Transpose the image according to its orientation tag (if available).

b) Add white background to image if it contains alpha transparency.

c) Crop uniformly coloured borders if applicable.

d) Convert image to grayscale.

e) Resize grayscale image to 32x32 pixels using bicubic interpolation.

5.4.5.2 An ISCC processor shall calculate the Image-Code as follows:

a) Apply discrete cosine transform to the 32x32 grayscale pixel matrix.

b) Calculate the median value of the upper left 8x8 pixels of the transformed matrix.

c) For each pixel of the upper-left 8x8 square set a 1-bit if the grayscale value is larger than the median and a 0-bit if it is smaller or equal to the median value.

d) The collected bits are the first 64 bits of the body of the Image-Code.

e) To extend the perceptual hash digest up to 256 bits repeat steps 2-3 for the top-right, bottom-left and bottom-right 8x8 squares in the given order.

5.4.6 Conformance

The normative behaviour of an ISCC processor in generating an Image–Code is specified only for the pre-processed 32x32 pixel grayscale input. An implementation of the Image-Code algorithm shall be regarded as conforming to the standard if it creates the same Image-Code as the reference implementation for the same 32x32 grayscale pixel values.

Implementers seeking to guarantee interoperability with each other in these circumstances should select the same tool for image pre-processing.

NOTE For further technical details, see source-code in modules “code_content_image.py” and “dct.py” of the reference implementation in: https://standards.iso.org/iso/24138/ed-1/en/.

5.5 Content-Code Subtype Audio

5.5.1 General

a) The Content-Code Subtype Audio (Audio-Code) shall be a similarity hash of the audio input.

b) The Audio-Code shall be robust against audio format conversion, compression, and minor edits.

5.5.2 Format

The Audio-Code shall have the data format illustrated in Figure 7.

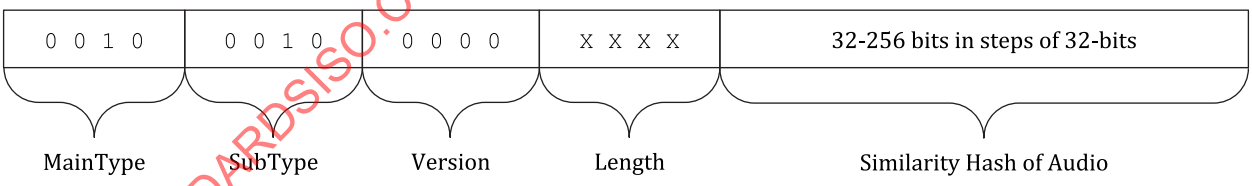

Figure 7 — Data format of the Audio-Code

EXAMPLE 1 64-bit Audio-Code in its canonical form:

ISCC:EIAWUJFCEZZOJYVD

EXAMPLE 2 256-bit Audio-Code in its canonical form:

ISCC:EIDWUJFCEZZOJYVDHJHIRB3KQSQCM2REUITDUTVAQNRGJIRENCCCULY

5.5.3 Inputs

The input for calculating the Audio-Code shall be the Chromaprint fingerprint (array of 32-bit signed integers) from the original audio data.

5.5.4 Outputs

Audio-Code processing shall generate the following ISCC metadata output elements:

a) iscc: the Audio-Code in its canonical form (required).

b) duration: duration of audio in seconds (optional).

c) Additional metadata extracted from the audio file (optional).

5.5.5 Processing

An ISCC processor shall calculate the Audio-Code as follows:

a) Calculate a 32-bit similarity hash from the input array.

b) Additionally, calculate 32-bit similarity hashes separately for each quarter of values from the input array and concatenate them to the result from step 1 in order to extend the similarity hash up to a total of 160 bits.

c) Sort the input array by value in ascending order and calculate similarity hashes separately for each third of the values and concatenate them to the result from step 2 in order to extend the similarity hash of the final ISCC-BODY of the Audio-Code up to the maximum of 256 bits.

5.5.6 Conformance

The normative behaviour of an ISCC processor in generating an Audio–Code is specified only for the Chromaprint input array. An implementation of the Audio-Code algorithm shall be regarded as conforming to this document if it creates the same Audio-Code as the reference implementation for the same Chromaprint array of 32-bit values.

NOTE For further technical details, see source-code in modules “code_content_audio.py” and “simhash.py” of the reference implementation in https://standards.iso.org/iso/24138/ed-1/en/.

5.6 Content-Code Subtype Video

5.6.1 General

a) The Content-Code Subtype Video (Video-Code) shall be a similarity hash of the input video.

b) The Video-Code shall be robust against format conversions, scaling, compression, changes of framerate and minor edits.

5.6.2 Format

The Video-Code shall have the data format illustrated in Figure 8.

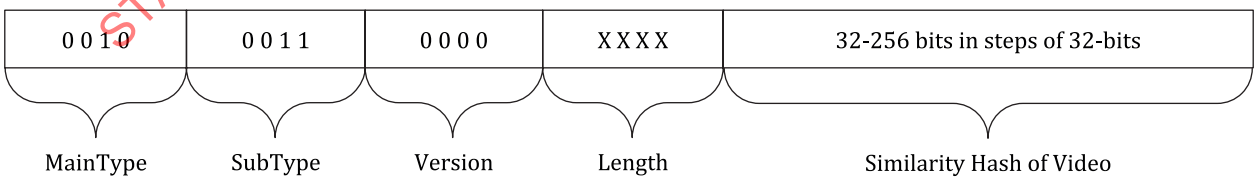

Figure 8 — Data format of the Video-Code

EXAMPLE 1 64-bit Video-Code in its canonical form:

ISCC:EMA7KERCWROEVL6F

EXAMPLE 2 256-bit Video-Code in its canonical form:

ISCC:EMD7KERCWROEVL6FU3SISZAZPJCBAZKXEZLZTSRQPGASTENCMSYFOAY

5.6.3 Inputs

The input for calculation of the Video-Code shall be the MPEG-7 video frame signatures, as specified in ISO/IEC 15938, from the original video data.

5.6.4 Outputs

Video-Code processing shall generate the following ISCC metadata output elements:

a) iscc: the Video-Code in its canonical form (required);

b) width: width of the original input video in number of pixels (optional);

c) height: height of the original input video in number of pixels (optional);

d) duration: duration of video in seconds (optional);

e) fps: frames per second of the original video (optional);

f) thumbnail: a thumbnail from the original video encoded as Data-URL (optional);

g) Additional metadata extracted from the video (optional).

5.6.5 Processing

5.6.5.1 An ISCC processor shall pre-process the video file input as follows:

a) Generate the MPEG-7 video signature at 5 frames per second.

5.6.5.2 An ISCC processor shall calculate the Video-Code as follows:

a) For each of the 380 integer values of the MPEG-7 frame signature calculate its sum over all frames.

b) Apply WTA hash algorithm to the resulting array of 380 integer values to calculate the ISCC-BODY of the Video-Code at the desired bit-length.

5.6.6 Conformance

The normative behaviour of an ISCC processor in generating a Video–Code is specified only for the MPEG-7 video frame signature input. An implementation of the Video-Code algorithm shall be regarded as conforming to the standard if it creates the same Video-Code as the reference implementation for the same MPEG-7 video frame signature input.

NOTE For further technical details, see source-code in modules “code_content_video.py” and “wtahash.py” of the reference implementation in https://standards.iso.org/iso/24138/ed-1/en/.

5.7 Content-Code Subtype Mixed

5.7.1 General

a) The Content-Code Subtype Mixed (Mixed-Code) shall be a similarity hash of a collection of assets of the same or different media types combined into a single multimedia file.

b) An ISCC processor that supports the creation of Mixed-Codes shall publicly document the supported file formats and the rules by which it divides the different parts of a multimedia file.

c) The Mixed-Code shall be robust against format conversions, scaling, compression, and minor edits of the individual parts of the multimedia file.

5.7.2 Format

The Mixed-Code shall have the data format illustrated in Figure 9:

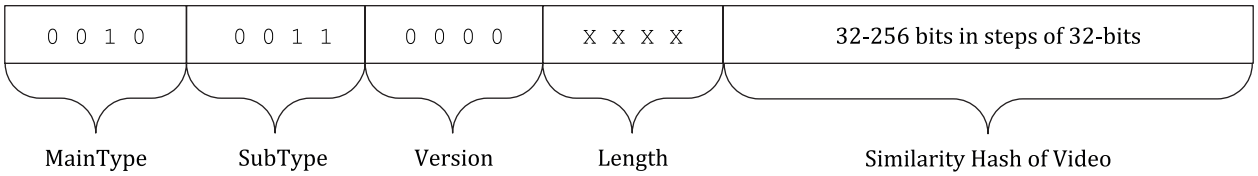

Figure 9 — Data format of the Mixed-Code

EXAMPLE 1 64-bit Mixed-Code in its canonical form:

ISCC:EQASD57JXX7U73P7

EXAMPLE 2 256-bit Mixed-Code in its canonical form:

ISCC:EQDSD57JXX7U73P7HPPH2P3U5OXZM7PL65T3HZ5JZ76H577P77NO5ZY

5.7.3 Inputs

a) The input for calculating the Mixed-Code shall be the Content-Codes of the individual parts of the multimedia file.

b) At least two Content-Codes shall be required as input to calculate a Mixed-Code.

5.7.4 Outputs

Mixed-Code processing shall generate the following ISCC metadata output elements:

a) iscc: the Mixed-Code in its canonical form (required);

b) parts: the list of Content–Codes used for calculating the Mixed-Code (recommended);

c) Additional metadata extracted from the multimedia file (optional).

5.7.5 Processing

5.7.5.1 An ISCC processor shall pre-process the multimedia file as follows:

a) Generate individual Content-Codes for each part of the multimedia file according to the specifications in 5.3, 5.4, 5.5 and 5.6.

5.7.5.2 An ISCC processor shall calculate the Mixed-Code as follows:

a) Create a byte sequence from each Content-Code retaining the first byte of the ISCC-HEADER concatenated with the bytes of the ISCC-BODY.

b) Apply the similarity hash to the list of byte sequences from step 1 to calculate the ISCC-BODY of the Mixed-Code.

5.7.6 Conformance

The normative behaviour of an ISCC processor in generating a Mixed-Code is specified only for Content-Code inputs. An implementation of the Mixed-Code algorithm shall be regarded as conforming to the standard if it creates the same Mixed-Code as the reference implementation for the same Content-Code inputs.

NOTE For further technical details, see source-code in modules “code_content_mixed.py” and “simhash.py” of the reference implementation in https://standards.iso.org/iso/24138/ed-1/en/.

5.8 Data-Code

5.8.1 General

a) The Data-Code shall be a similarity hash for any kind of data regardless of its media type.

b) The Data-Code shall cluster digital assets that have near-identical data.

c) Small differences (as a proportion of the whole) in referent data shall yield identical Data-Codes.

d) More significant differences in referent data shall produce similar Data-Codes that can be compared against each other to estimate the data-similarity of the referents.

e) The Data-Code shall be resistant to data shifting and reordering sequences of data within referent data.

NOTE Changes of the Data-Code do not reflect semantic or syntactic changes of the content.

5.8.2 Format

The Data-Code shall have the data format illustrated in Figure 10:

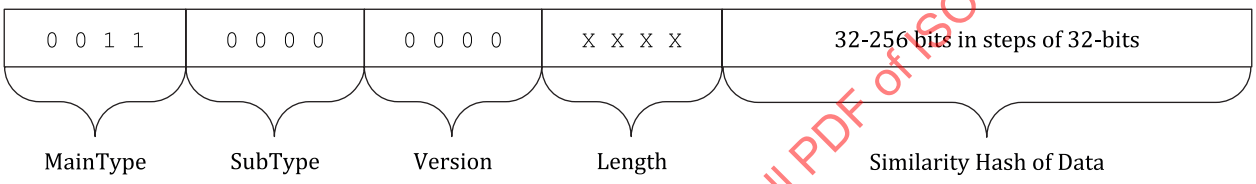

Figure 10 — Data format of the Data-Code

EXAMPLE 1 64-bit Data-Code in its canonical form:

ISCC:GAAWAIBQLNWP7X32

EXAMPLE 2 256-bit Data-Code in its canonical form:

ISCC:GADWAIBQLNWP7X32J3INMAMDUJ4QMN67BBQKVTVZIWHXQ7QJIKHYTBY

5.8.3 Inputs

The input for calculating the Data-Code shall be the bytes of a file, without reference to their meaning or structure.

5.8.4 Outputs

Data-Code processing shall generate the following output elements:

— iscc: the Data-Code in its canonical form (required).

5.8.5 Processing

An ISCC processor shall calculate the Data-Code as follows:

a) Split the data into variable sized chunks with an average chunk size of 1024 bytes using the content defined chunking (CDC) algorithm.

b) Calculate the 32-bit integer hash digest of each chunk using the XXH32 algorithm.

c) Apply the minhash algorithm to the array of 32-bit integers to calculate the ISCC-BODY of the Data-Code with appropriate length.

NOTE For further technical details, see source-code in modules “code_data.py” and “minhash.py” of the reference implementation in https://standards.iso.org/iso/24138/ed-1/en/.

5.8.6 Conformance

An implementation of the Data-Code algorithm shall be regarded as conforming to this document if it creates the same Data-Code as the reference implementation for the same data input.

NOTE The ISCC reference implementation uses the open source XXHASH library[11] for XXH32 chunk hashing and appropriate use of this software will generate the same codes as the reference implementation.

5.9 Instance-Code

5.9.1 General

a) The Instance-Code shall be a checksum for any kind of data regardless of its media type.

b) The Instance-Code shall match identical files and indicate data transmission errors or data manipulation.

c) The Instance-Code shall be the prefix of a Merkle root of a Merkle tree as illustrated in Figure 11.

d) The cryptographic hash algorithm used for Instance-Code calculation shall be updated if a security weakness is discovered.

e) An update shall be indicated by a higher version number of the Instance-Code header.

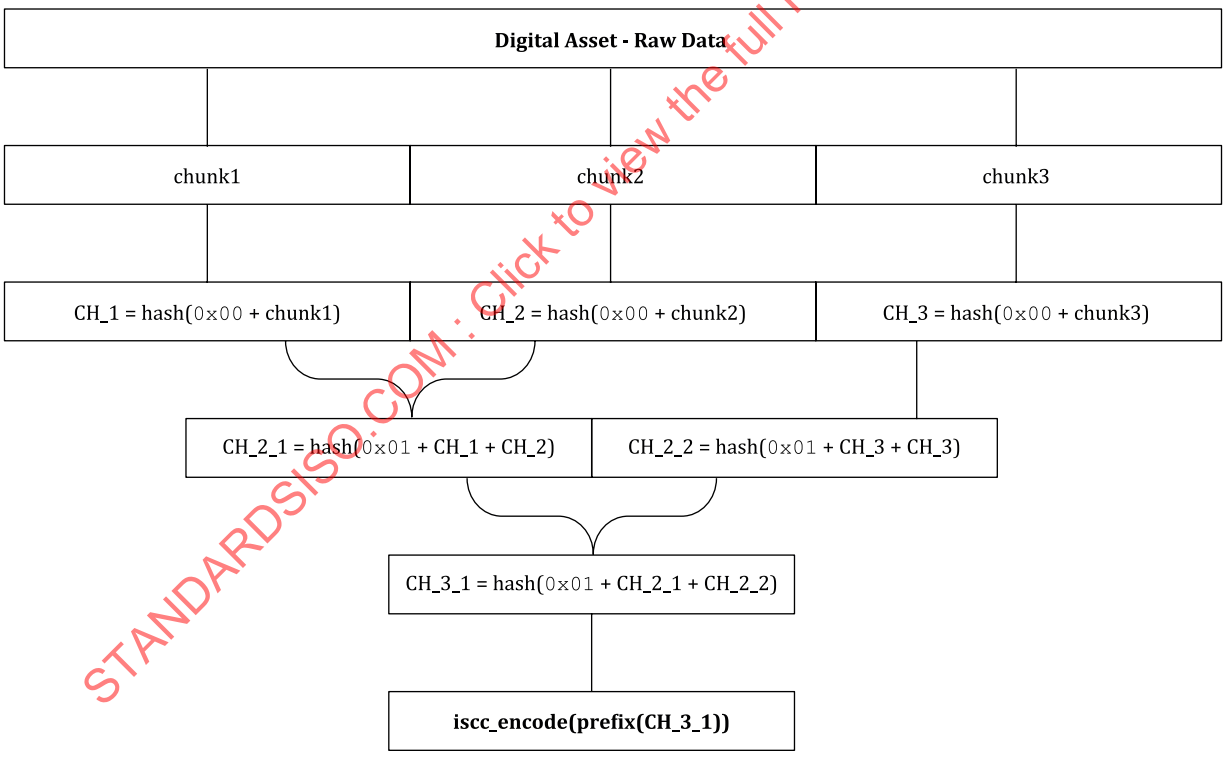

Figure 11 — Example of a Merkle tree

NOTE 1 Depending on security and uniqueness requirements, applications can generate and store Instance-Codes up to a length of 256 bits in steps of 32 bits. Longer variants of an Instance-Code are extensions of the shorter versions (which function only as checksums) and remain compatible while improving the security of the hash algorithm.

NOTE 2 The use of tree-based hashing supports efficient containment-proofs, verified streaming and verification of data from trusted and untrusted sources. A data delivery application can provide cryptographic proofs for partial data and streams such that the receiving application can incrementally verify chunks of data against a given Instance-Code.

5.9.2 Format

The Instance-Code shall have the data format illustrated in Figure 12.

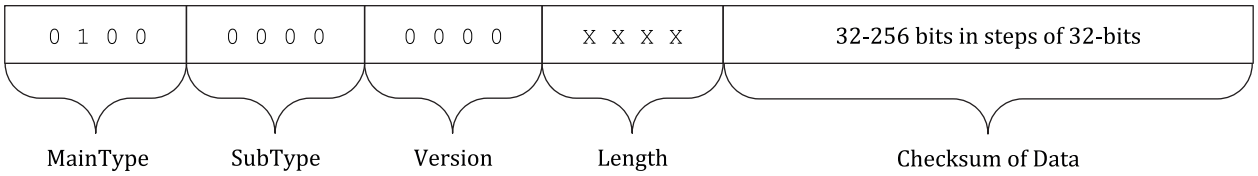

Figure 12 — Data format of the Instance-Code

EXAMPLE 1 64-bit Instance-Code in its canonical form:

ISCC:IAAZ3NGA3HTIYUQD

EXAMPLE 2 256-bit Instance-Code in its canonical form:

ISCC:IADZ3NGA3HTIYUQD3SGC737FF6S5KRTRXY5DEU7ANCEMVTT4MDS2OQY

5.9.3 Inputs

The input for calculating the Instance-Code shall be the bytes of a file, without reference to their meaning or structure.

5.9.4 Outputs

Instance-Code processing shall generate the following output elements:

a) iscc: the Instance-Code in its canonical form (required);

b) datahash: a cryptographic hash of the input file encoded as multihash (optional);

c) filesize: the size of the input file in bytes (optional).

5.9.5 Processing

An ISCC processor shall calculate the Instance-Code as follows:

a) Apply the tree-based cryptographic hash function to the file input using the appropriate number of bytes for the output digest.

b) Use the hash digest of output as the ISCC-BODY of the Instance-Code.

For further technical details, see source-code in modules “code_instance.py” of the reference implementation in the supplementary electronic inserts.

NOTE The Merkle tree chunk size is defined by the BLAKE3[10] algorithm to be 1024 bytes.

5.9.6 Conformance

An implementation of the Instance-Code algorithm shall be regarded as conforming to this document if it creates the same Instance-Code as the reference implementation for the same data input.

NOTE The ISCC reference implementation uses the open source BLAKE3[10] library for merkle tree hashing and appropriate use of this software will generate the same codes as the reference implementation.

6 ISCC-CODE

6.1 General

An ISCC-CODE shall be an ordered sequence of two or more headerless ISCC-UNITs of different MainTypes derived from one referent prefixed with a common header.

6.2 Purpose

The ISCC-CODE shall support identification, clustering, discovery and matching of files based on their metadata, content, data similarity, and where appropriate it shall be used together with other identifiers in accordance with the principles outlined in Annex A.

6.3 Format

6.3.1 General

The ISCC-CODE shall have the data format illustrated in Figure 13.

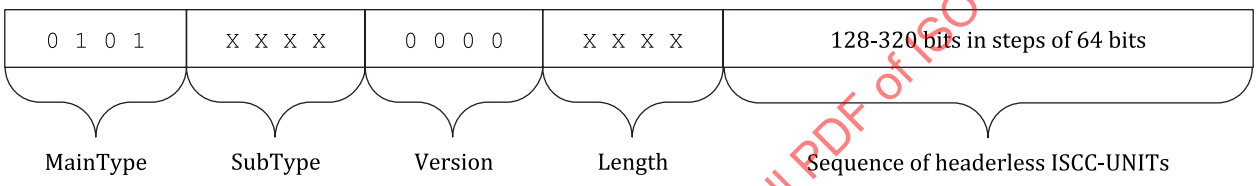

Figure 13 — Data format of the ISCC-CODE

EXAMPLE 1

128-bit ISCC-CODE in its canonical form, that includes a Data-Code and Instance-Code:

ISCC:KUAIFYXGML3SRNH25MIWPM3HVHBXQ

EXAMPLE 2

256-bit ISCC-CODE in its canonical form, that includes a Meta-Code, Text-Code, Data-Code, and an Instance-Code:

ISCC:KAC6HZYGQLBASTFMBJOS6NDLVKKFLAXC4ZRPOKFU7LVRCZ5TM6U4G6A

6.3.2 SubTypes for ISCC-CODEs

a) If an ISCC-UNIT of type CONTENT is sequenced into an ISCC-CODE, the SubType of the ISCC-CODE shall be that of the Content-Code (see 4.2.3).

b) If the ISCC-CODE is composed only of a sequence of the types META, DATA, and INSTANCE, the SubType shall be NONE.

c) If the ISCC-CODE is composed only of a sequence of types DATA and INSTANCE, the SubType shall be SUM.

6.3.3 Length and composition of ISCC-CODEs

a) The length in bits of the ISCC–BODY of an ISCC-CODE shall be calculated as the number of data bits set in the Length field of the header times 64 plus 128 bits.

b) The ISCC-UNITs composed into an ISCC-CODE shall be ordered as follows: META, SEMANTIC, CONTENT, DATA, INSTANCE.

c) The data bits of the Length field shall be the bits following the prefix bit(s) and they shall encode the composition of the ISCC-BODY of an ISCC-CODE as follows:

   1) The first data bit shall signify the presence of a Meta-Code.

   2) The second data bit shall signify the presence of a Semantic-Code.

   3) The third data bit shall signify the presence of a Content-Code.

6.4 Inputs

a) The input for calculating an ISCC-CODE shall be a collection of ISCC-UNITs.

b) The input shall include at least a Data-Code and an Instance-Code with a minimum of 64 bits each.

c) The input shall include at most one ISCC-UNIT of each MainType.

d) If both a Semantic-Code and a Content-Code are given as input, they shall be of the same SubType.

6.5 Outputs

ISCC-CODE processing shall generate the following output elements:

a) iscc: the ISCC-CODE in its canonical form (required);

b) filename: the name of the input file (optional);

c) any other elements collected during processing of the individual ISCC-UNITs (optional).

6.6 Processing

An ISCC processor shall compose an ISCC-CODE as follows.

a) Sort the ISCC-UNITs according to their predefined order (META, SEMANTIC, CONTENT, DATA, INSTANCE).

b) Decode the ISCC-UNITs to binary and if supplied SubTypes of the Semantic-Code and Content-Code are different, halt.

c) Remove the headers of the decoded ISCC-UNITs.

d) Truncate the ISCC-UNITs by keeping only the first 64 bits.

e) Concatenate the headerless and truncated ISCC-UNITs in sorted order to construct the final ISCC-BODY of the ISCC-CODE.

f) Prefix the ISCC-BODY with the appropriate ISCC-HEADER and encode the result to its canonical form.

For further details, see source-code in the module “iscc_code.py” of the reference implementation in https://standards.iso.org/iso/24138/ed-1/en/.

6.7 Comparing ISCC-CODEs

To measure the similarity of two ISCC-CODEs, check if the Instance-Codes are identical. Calculate the binary hamming distance of the ISCC-BODYs of the other ISCC-UNITs with the same MainType and SubType. Lower values of the hamming distance indicate higher probability of similarity. Higher values of the hamming distance indicate decreasing similarity. The threshold indicating identity will vary according to the MainType and the application.

For further details, see source-code of the function “iscc_compare” in the module “utils.py” of the reference implementation in the electronic inserts.

6.8 Conformance

An implementation of the ISCC-CODE algorithm shall be regarded as conforming to this document if it creates the same ISCC-CODE as the reference implementation for the same data input.