
**Information technology — Coding of
audio-visual objects —**

**Part 12:
ISO base media file format**

*Technologies de l'information — Codage des objets audiovisuels —
Partie 12: Format ISO de base pour les fichiers médias*





COPYRIGHT PROTECTED DOCUMENT

© ISO/IEC 2022

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

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

Published in Switzerland

Contents

Page

Foreword	x
Introduction	xi
1 Scope	1
2 Normative references	1
3 Terms, definitions and abbreviated terms	2
3.1 Terms and definitions	2
3.2 Abbreviated terms	7
4 Object-structured file organization	7
4.1 File structure	7
4.2 Object structure	7
4.2.1 Object syntax conventions	7
4.2.2 Object definitions	8
4.2.3 Extensibility of object definitions	9
4.3 File-type box	10
4.3.1 Definition	10
4.3.2 Syntax	10
4.3.3 Semantics	11
4.4 Extended type box	11
4.4.1 Definition	11
4.4.2 Syntax	11
4.4.3 Semantics	11
5 Structure of this document	12
6 ISO base media file organization	12
6.1 Files, segments, and streams	12
6.2 Presentation structure	13
6.2.1 Object structure of a presentation	13
6.2.2 Meta data and media data	13
6.3 Structure-data (objects)	13
6.3.1 Box	13
6.3.2 Data types and fields	13
6.3.3 URIs as type indicators	14
6.3.4 Box order	15
6.4 Time structure overview	18
6.5 Identifiers	19
6.6 Brand identification	19
6.7 Uniform resource locators (URLs)	19
7 Streaming support	19
8 Box structures	19
8.1 File structure and general boxes	19
8.1.1 Media data box	19
8.1.2 Free space box	20
8.1.3 Progressive download information box	20
8.1.4 Identified media data box	21
8.2 Movie structure	21
8.2.1 Movie box	21
8.2.2 Movie header box	21
8.3 Track structure	23
8.3.1 Track box	23
8.3.2 Track header box	23
8.3.3 Track reference box	26
8.3.4 Track group box	28

8.3.5	Track type box	29
8.4	Track media structure	30
8.4.1	Media box	30
8.4.2	Media header box	30
8.4.3	Handler reference box	31
8.4.4	Media information box	32
8.4.5	Media information header boxes	32
8.4.6	Extended language tag	32
8.5	Sample tables	33
8.5.1	Sample table box	33
8.5.2	Sample description box	34
8.5.3	Degradation priority box	36
8.5.4	Sample scale box	36
8.6	Track time structures	36
8.6.1	Time to sample boxes	36
8.6.2	Sync sample box	41
8.6.3	Shadow sync	42
8.6.4	Independent and disposable samples box	43
8.6.5	Edit box	45
8.6.6	Edit list box	45
8.7	Track data layout structures	48
8.7.1	Data information box	48
8.7.2	Data reference box	48
8.7.3	Sample size boxes	50
8.7.4	Sample to chunk box	51
8.7.5	Chunk offset box	52
8.7.6	Padding bits box	52
8.7.7	Sub-sample information box	53
8.7.8	Sample auxiliary information sizes box	54
8.7.9	Sample auxiliary information offsets box	56
8.8	Movie fragments	57
8.8.1	Movie extends box	57
8.8.2	Movie extends header box	58
8.8.3	Track extends box	58
8.8.4	Movie fragment box	59
8.8.5	Movie fragment header box	60
8.8.6	Track fragment box	60
8.8.7	Track fragment header box	60
8.8.8	Track fragment run box	62
8.8.9	Movie fragment random access box	63
8.8.10	Track fragment random access box	64
8.8.11	Movie fragment random access offset box	65
8.8.12	Track fragment decode time box	65
8.8.13	Level assignment box	66
8.8.14	Sample auxiliary information in movie fragments	68
8.8.15	Track Extension Properties box	68
8.8.16	Alternative startup sequence properties box	68
8.8.17	Metadata and user data in movie fragments	69
8.9	Sample group structures	70
8.9.1	Overview	70
8.9.2	Sample to group box	70
8.9.3	Sample group description box	72
8.9.4	Representation of group structures in movie fragments	74
8.9.5	Compact sample to group box	75
8.10	User data	77
8.10.1	User data box	77
8.10.2	Copyright box	77
8.10.3	Track selection box	78

8.10.4	Track kind.....	79
8.11	Metadata support.....	80
8.11.1	MetaBox.....	80
8.11.2	XML boxes.....	81
8.11.3	Item location box.....	81
8.11.4	Primary item box.....	84
8.11.5	Item protection box.....	85
8.11.6	Item information box.....	85
8.11.7	Additional metadata container box.....	87
8.11.8	Metabox Relation box.....	87
8.11.9	URL forms for MetaBoxes.....	88
8.11.10	Static metadata.....	88
8.11.11	Item data box.....	89
8.11.12	Item reference box.....	89
8.11.13	Auxiliary video metadata.....	90
8.11.14	Item properties box.....	90
8.11.15	Brand item property.....	92
8.12	Support for protected streams.....	93
8.12.1	Overview.....	93
8.12.2	Protection scheme information box.....	94
8.12.3	Original format box.....	94
8.12.4	IPMPInfoBox.....	95
8.12.5	IPMP control box.....	95
8.12.6	Scheme type box.....	95
8.12.7	Scheme information box.....	95
8.12.8	Scramble Scheme Information Box.....	96
8.13	File delivery format support.....	96
8.13.1	Overview.....	96
8.13.2	FD item information box.....	97
8.13.3	File partition box.....	97
8.13.4	FEC reservoir box.....	99
8.13.5	FD session group box.....	99
8.13.6	Group ID to name box.....	100
8.13.7	File reservoir box.....	101
8.14	Sub tracks.....	101
8.14.1	Overview.....	101
8.14.2	Backward compatibility.....	102
8.14.3	Sub track box.....	102
8.14.4	Sub track information box.....	102
8.14.5	Sub track definition box.....	103
8.14.6	Sub track sample group box.....	104
8.15	Post-decoder requirements on media.....	104
8.15.1	General.....	104
8.15.2	Restricted sample entry transformation.....	105
8.15.3	Restricted scheme information box.....	105
8.15.4	Scheme for stereoscopic video arrangements.....	106
8.15.5	Compatible scheme type box.....	108
8.16	Segments.....	108
8.16.1	Overview.....	108
8.16.2	Segment type box.....	108
8.16.3	Segment index box.....	109
8.16.4	Subsegment index box.....	112
8.16.5	Producer reference time box.....	114
8.17	Support for incomplete tracks.....	115
8.17.1	General.....	115
8.17.2	Transformation.....	116
8.17.3	Complete track information box.....	116
8.18	Entity grouping.....	117

8.18.1	General	117
8.18.2	Groups list box	117
8.18.3	Entity to group box	117
8.19	Compressed boxes	118
8.19.1	Overview and processing	118
8.19.2	Processing model	119
8.19.3	General syntax	120
8.19.4	General semantics	120
8.19.5	Original file-type box	120
8.19.6	Compressed movie box	121
8.19.7	Compressed movie fragment box	121
8.19.8	Compressed segment index box	121
8.19.9	Compressed subsegment index box	122
9	Hint track formats	122
9.1	RTP and SRTP hint track format	122
9.1.1	Overview	122
9.1.2	Sample description format	123
9.1.3	Sample format	124
9.1.4	SDP information	127
9.1.5	Statistical information	127
9.2	ALC/LCT and FLUTE hint track format	128
9.2.1	Overview	128
9.2.2	Design principles	129
9.2.3	Sample description format	130
9.2.4	Sample format	130
9.3	MPEG-2 transport hint track format	133
9.3.1	Overview	133
9.3.2	Design principles	134
9.3.3	Sample description format	135
9.3.4	Sample format	137
9.3.5	Protected MPEG 2 transport stream hint track	139
9.4	RTP, RTCP, SRTP and SRTCP reception hint tracks	140
9.4.1	RTP reception hint track	140
9.4.2	RTCP reception hint track	143
9.4.3	SRTP reception hint track	144
9.4.4	SRTCP reception hint tracks	146
9.4.5	Protected RTP reception hint track	147
9.4.6	Recording procedure	147
9.4.7	Parsing procedure	147
10	Sample groups	147
10.1	Random access recovery points	147
10.1.1	Definition	147
10.1.2	Syntax	148
10.1.3	Semantics	148
10.2	Rate share groups	148
10.2.1	Overview	148
10.2.2	Rate share sample group entry	149
10.2.3	Relationship between tracks	150
10.2.4	Bitrate allocation	151
10.3	Alternative startup sequences	151
10.3.1	Definition	151
10.3.2	Syntax	152
10.3.3	Semantics	152
10.3.4	Examples	152
10.4	Random access point (RAP) sample group	154
10.4.1	Definition	154
10.4.2	Syntax	154

10.4.3	Semantics	154
10.5	Temporal level sample group	154
10.5.1	Definition	154
10.5.2	Syntax	155
10.5.3	Semantics	155
10.6	Stream access point sample group	155
10.6.1	Definition	155
10.6.2	Syntax	155
10.6.3	Semantics	155
10.7	Sample-to-item sample group	156
10.7.1	Definition	156
10.7.2	Syntax	156
10.7.3	Semantics	156
10.8	Dependent random access point (DRAP) sample group	156
10.8.1	Definition	156
10.8.2	Syntax	157
10.8.3	Semantics	157
10.9	Pixel Aspect Ratio Sample Grouping	157
10.9.1	Definition	157
10.9.2	Syntax	157
10.9.3	Semantics	157
10.10	Clean Aperture Sample Grouping	157
10.10.1	Definition	157
10.10.2	Syntax	158
10.10.3	Semantics	158
11	Derived file formats	158
12	Media-specific definitions	159
12.1	Video media	159
12.1.1	Media handler	159
12.1.2	Video media header	159
12.1.3	Sample entry	159
12.1.4	Pixel aspect ratio and clean aperture	160
12.1.5	Colour information	162
12.1.6	Content light level	163
12.1.7	Mastering display colour volume	163
12.1.8	Content colour volume	163
12.1.9	Ambient viewing environment	164
12.2	Audio media	164
12.2.1	Media handler	164
12.2.2	Sound media header	164
12.2.3	Sample entry	165
12.2.4	Channel layout	167
12.2.5	Downmix instructions	169
12.2.6	DRC information	172
12.2.7	Audio stream loudness	173
12.3	Metadata media	175
12.3.1	Media handler	175
12.3.2	Media header	175
12.3.3	Sample entry	175
12.4	Hint media	177
12.4.1	Overview	177
12.4.2	Media handler	178
12.4.3	Hint media header	178
12.4.4	Sample entry	178
12.5	Text media	179
12.5.1	Media handler	179
12.5.2	Media header	179

12.5.3	Sample entry	179
12.6	Subtitle media	179
12.6.1	Media handler	179
12.6.2	Subtitle media header	179
12.6.3	Sample entry	180
12.7	Font media	181
12.7.1	Media handler	181
12.7.2	Media header	181
12.7.3	Sample entry	181
12.8	Transformed media	181
12.8.1	General	181
12.8.2	Multiple transformations for a single transformed media track	181
12.8.3	Determining the untransformed sample entry type	182
12.8.4	The 'codecs' MIME parameter for a transformed media track	182
12.9	Multiplexed timed metadata tracks	182
12.9.1	General	182
12.9.2	Overall design	182
12.9.3	Sample format	183
12.9.4	Sample entry format	183
12.9.5	Defined formats	186
12.10	Volumetric visual media	187
12.10.1	Media handler	187
12.10.2	Media header	187
12.10.3	Sample entry	187
12.10.4	Sample format	188
12.11	Haptic media	188
12.11.1	Media handler	188
12.11.2	Media header	188
12.11.3	Sample entry	188
12.11.4	Sample format	188
Annex A (informative) Background and tutorial		189
Annex B (informative) Guidance on deriving from this document		197
Annex C (normative) Fragment identifiers for ISO base media resources		206
Annex D (informative) Management of extension code-points		207
Annex E (normative) File format brands		209
Annex F (normative) MIME type registration of segments		220
Annex G (informative) URI-labelled metadata forms		221
Annex H (informative) Processing of RTP streams and reception hint tracks		223
Annex I (normative) Stream access points		240
Annex J (informative) Segment index examples		243
Annex K (normative) Use of IETF RFC 6381 for ISO BMFF files		246
Bibliography		249

Foreword

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

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

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

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

This seventh edition cancels and replaces the sixth edition (ISO/IEC 14496-12:2020), which has been technically revised.

The main changes are as follows:

- re-organization of all the introductory material, such that the material needed by a specific audience or applying to specific aspects is brought together;
- terminology with respect to timing is more consistent and simpler;
- replacing the word 'metadata' as describing the structural data, as the word metadata is also used with another meaning, and the dual use was confusing;
- providing better wording for the `TrackHeader` flags;
- other minor editorial improvements.

A list of all parts in the ISO/IEC 14496 series can be found on the ISO and IEC websites.

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

Introduction

The ISO base media file format is designed to contain timed media information for a presentation in a flexible, extensible format that facilitates interchange, management, editing, and presentation of the media. This presentation may be 'local' to the system containing the presentation, or may be via a network or other stream delivery mechanism.

The file structure is object-oriented; a file can be decomposed into constituent objects very simply, and the structure of the objects inferred directly from their type.

The file format is designed to be independent of any particular network protocol while enabling efficient support for them in general.

The ISO base media file format is a base format for media file formats.

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 patents.

ISO and IEC take no position concerning the evidence, validity and scope of these patent rights

The holders of these patent rights have assured ISO and IEC that they are willing to negotiate licences under reasonable and non-discriminatory terms and conditions with applicants throughout the world. In this respect, the statements of the holders of these patent rights are 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.

Information technology — Coding of audio-visual objects —

Part 12: ISO base media file format

1 Scope

This document specifies the ISO base media file format, which is a general format forming the basis for a number of other more specific file formats. This format contains the timing, structure, and media information for timed sequences of media data, such as audio-visual presentations.

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 639-2, *Codes for the representation of names of languages — Part 2: Alpha-3 code*

ITU-T X.667/ ISO/IEC 9834-8, *Information technology – Procedures for the operation of object identifier registration authorities – Part 8: Generation of universally unique identifiers (UUIDs) and their use in object identifiers*

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

ISO/IEC 13818-2:2013, *Information technology — Generic coding of moving pictures and associated audio information — Part 2: Video*

ISO/IEC 14496-1, *Information technology — Coding of audio-visual objects — Part 1: Systems*

ISO/IEC 14496-10:2014, *Information technology — Coding of audio-visual objects — Part 10: Advanced Video Coding*

ISO 15076-1, *Image technology colour management — Architecture, profile format and data structure — Part 1: Based on ICC.1:2010*

ISO/IEC 15938-1, *Information technology — Multimedia content description interface — Part 1: Systems*

ISO/IEC 23001-1, *Information technology — MPEG systems technologies — Part 1: Binary MPEG format for XML*

ISO/IEC 23002-3, *Information technology — MPEG video technologies — Part 3: Representation of auxiliary video and supplemental information*

ISO/IEC 23003-4, *Information technology — MPEG audio technologies — Part 4: Dynamic range control*

ITU-T H.265 / ISO/IEC 23008-2, *Information technology — High efficiency coding and media delivery in heterogeneous environments — Part 2: High efficiency video coding*

ISO/IEC 23091-2, *Information technology — Coding-independent code points — Part 2: Video*

ISO/IEC 23091-3, *Information technology — Coding-independent code points — Part 3: Audio*

IETF RFC 1951, *DEFLATE Compressed Data Format Specification version 1.3*

IETF RFC 2045, *Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies*

IETF RFC 2046, *Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types*

IETF RFC 3629, *UTF-8, a transformation format of ISO 10646*

IETF RFC 3711:2004, *The Secure Real-time Transport Protocol (SRTP)*

IETF RFC 5052, *Forward Error Correction (FEC) Building Block*

IETF RFC 5905, *Network Time Protocol Version 4: Protocol and Algorithms Specification*

ITU-R TF.460-6:2002, *Standard-frequency and time-signal emissions*

ITU-R BS.1770-4, *Algorithms to measure audio programme loudness and true-peak audio level*

IETF BCP 47, *Tags for Identifying Languages*

IETF RFC 4122, *A Universally Unique IDentifier (UUID) URN Namespace*

IETF RFC 3061, *A URN Namespace of Object Identifiers*

W3C Recommendation, *Extensible Markup Language (XML) 1.0 (Fifth Edition)*, 26 November 2008, <https://www.w3.org/TR/2008/REC-xml-20081126/>

3 Terms, definitions and abbreviated terms

3.1 Terms and definitions

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

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

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

3.1.1

box

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

Note 1 to entry: Called 'atom' in some specifications, including the first definition of MP4

3.1.2

chunk

contiguous set of samples for one track

3.1.3

clean aperture

part of a decoded video image from which undesirable pixels introduced for coding purposes such as having integer number of coding blocks have been removed for presentation

3.1.4

container box

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

Note 1 to entry: Container boxes are normally not derived from FullBox

3.1.5

file level

byte position in an ISO base media file not contained in a Box structure

3.1.6**full aperture**

decoded video image as output by the decoder which may contain undesirable pixels for presentation

3.1.7**hint track**

special track which does not contain media data, but instead contains instructions for packaging one or more tracks into a streaming channel

3.1.8**hinter**

tool that is run on a file containing only media, to add one or more hint tracks to the file and so facilitate streaming

3.1.9**index file**

ISO base media file containing only `SegmentIndexBox`

3.1.10**ISO base media file**

file conforming to the file format described in this document (either a movie file, a metadata file, a segment file or an index file)

3.1.11**item**

data which does not require timed processing, as opposed to sample data

3.1.12**leading sample**

sample associated with a random access point (RAP) that precedes the RAP in composition order and immediately follows the RAP or another leading sample in decoding order, and which possibly cannot be correctly decoded when decoding starts from the RAP

3.1.13**leaf subsegment**

subsegment that does not contain any indexing information that would enable its further division into subsegments

3.1.14**mod**

modulo operator: $(x \bmod y) = x - y \text{ floor } (x/y)$

3.1.15**media data box**

box which can hold the actual media data for a presentation (`'mdat'`)

3.1.16**metadata file**

ISO base media file containing a top-level `MetaBox`

Note 1 to entry: A Movie File may also be a Metadata File, and vice-versa.

3.1.17**movie box**

container box whose sub-boxes define the structure-data for a presentation (`'moov'`)

3.1.18**movie file**

ISO base media file containing a `MovieBox`

3.1.19

movie fragment

fragment of the information contained in a `MovieBox`, defined by a `MovieFragmentBox` and its contents

3.1.20

structure-data

data that provides the location, size, timing, and characteristics of the media data (e.g. the coded frames of audio and video)

3.1.21

movie-fragment relative addressing

signalling of offsets for media data in movie fragments that is relative to the start of those movie fragments, specifically setting the flags `base-data-offset-present` to 0 and `default-base-is-moof` to 1 in `TrackFragmentHeaderBoxes`

Note 1 to entry: Setting the `default-base-is-moof` flag to 1 is only relevant for movie fragments that contain more than one track run (either in the same or several tracks).

3.1.22

open random access point

sample after which all samples in composition order can be correctly decoded, but some samples following the random access point in decoding order and preceding the random access point in composition order need not be correctly decodable

Note 1 to entry: For example, an intra picture starting an open group of pictures can be followed in decoding order by (bi-)predicted pictures that however precede the intra picture in composition order; though they possibly cannot be correctly decoded if the decoding starts from the intra picture, they are not needed.

3.1.23

pixel aspect ratio

scaling required to be applied to the output pixel of a decoder to produce a non-distorted image

Note 1 to entry: The term "Sample Aspect Ratio" is sometimes used for this term, but "sample" in this standard has a specific meaning.

3.1.24

presentation

one or more motion sequences, possibly combined with audio

3.1.25

presentation time

timeline of a track that aligns with the timelines of other tracks established by an explicit or implied edit list applied to the composition timestamps

3.1.26

timeline

monotonic linear representation of times with respect to a zero origin point

3.1.27

timescale

number of timestamp values that represent a duration of one second

3.1.28

timestamp

integer coded value representing an instant of time on an associated timeline

3.1.29**decoding timestamp**

timestamp on the media timeline such that samples are decoded in decoding timestamp order

Note 1 to entry: The decoding timestamps primarily define the required decoding order; systems may decode at a time of their choosing; in the case of signed composition offsets, decoding timestamps do not necessarily precede composition timestamps.

3.1.30**composition timestamp**

timestamp on the media timeline such that samples are presented in composition timestamp order and that establishes their relative composition timing

3.1.31**sample duration**

difference between the decoding timestamp of the following sample (when known) and the decoding timestamp of this sample

3.1.32**random access point****RAP**

sample in a track that starts at the ISAU of a SAP of type 1 or 2 or 3; informally, a sample, from which when decoding starts, the sample itself and all samples following in composition order can be correctly decoded

Note 1 to entry: SAP types are defined in [Annex I](#).

3.1.33**random access recovery point**

sample in a track with presentation time equal to the TSAP of a SAP of type 4; informally, a sample, that can be correctly decoded after having decoded a number of samples that is before this sample in decoding order, sometimes known as gradual decoding refresh

Note 1 to entry: SAP types are defined in [Annex I](#).

3.1.34**sample**

all the data associated with a single time

Note 1 to entry: No two samples within a track can share the same decoding timestamp; no two samples can share the same composition timestamp.

Note 2 to entry: In non-hint tracks, a sample is, for example, an individual frame of video, a series of video frames in decoding order, or a compressed section of audio in decoding order; in hint tracks, a sample defines the formation of one or more streaming packets.

3.1.35**sample description**

structure which defines and describes the format of some number of samples in a track

3.1.36**sample entry type**

four-character code that is either a format value of a `SampleEntry` directly contained in `SampleDescriptionBox` or a `data_format` value of `OriginalFormatBox`

3.1.37**untransformed sample entry type**

sample entry type of the track that would apply if no transformations had been performed to a transformed media track

Note 1 to entry: This is the *sample entry type* that would be the format value in a `SampleEntry` directly contained in the `SampleDescriptionBox`.

3.1.38

sample number

ordinal index number of a given sample where the first sample has sample number 1

3.1.39

sample table

packed directory for the timing and physical layout of the samples in a track

3.1.40

sync sample

sample in a track that starts at the ISAU of a SAP of type 1 or 2

Note 1 to entry: SAP types are defined in [Annex I](#).

Note 2 to entry: Informally, a media sample that starts a new independent sequence of samples; if decoding starts at the sync sample, it and succeeding samples in decoding order can all be correctly decoded, and the resulting set of decoded samples forms the correct presentation of the media starting at the decoded sample that has the earliest composition time; a media format may provide a more precise definition of a sync sample for that format.

3.1.41

segment

portion of movie file, consisting of either (a) a `MovieBox`, with its associated media data (if any) and other associated boxes or (b) one or more `MovieFragmentBoxes`, with their associated media data, and other associated boxes

Note 1 to entry: The associated media data can be found by following byte offsets, but the process of finding associated boxes is not given in this standard and may be derived by other specifications.

3.1.42

segment file

ISO base media file containing one or more segment(s)

3.1.43

subsegment

time interval of a segment formed from `MovieFragmentBoxes`, that is also a valid segment

3.1.44

thumbnail image

smaller-resolution representation of an image

3.1.45

top-level box

box contained at file level

3.1.46

track

timed sequence of related samples (q.v.) in an ISO base media file

Note 1 to entry: For media data, a track corresponds to a sequence of images or sampled audio; for hint tracks, a track corresponds to a streaming channel.

3.1.47

haptic media

timed tactile signals to be presented as part of the media presentation

3.1.48

volumetric visual media

timed visual media defining a visual coding in a three-dimensional space

Note 1 to entry: In contrast to video media, which defines a planar coding.

3.2 Abbreviated terms

ALC	asynchronous layered coding
AVC	advanced video coding
FD	file delivery
FDT	file delivery table
FEC	forward error correction
FLUTE	file delivery over unidirectional transport
IANA	internet assigned numbers authority
IETF	internet engineering task force
LCT	layered coding transport
MBMS	multimedia broadcast/multicast service
MIME	multipurpose internet mail extensions (as defined in IETF RFC 2045 and IETF RFC 2046)
MVC	multiview video coding
SVC	scalable video coding
UUID	universally unique identifier (as defined in IETF RFC 4122 and ISO/IEC 9834-8)

4 Object-structured file organization

4.1 File structure

Files are formed as a series of objects, called boxes in this document. All data is contained in boxes; there is no other data within the file. This includes any initial signature required by the specific file format.

All object-structured files conformant to [Clause 4](#) (all object-structured files) shall contain a `FileTypeBox`.

In this document, top-level boxes (boxes not contained in other boxes) are indicated as being at ‘file’ level, with the notation “Container: File”.

4.2 Object structure

4.2.1 Object syntax conventions

The definitions of objects are given in the syntax description language (SDL) defined in ISO/IEC 14496-1.

NOTE Comments in the code fragments in this document are informative.

The fields in the objects are stored with the most significant byte first, commonly known as network byte order or big-endian format. When fields smaller than a byte are defined, or fields span a byte boundary, the bits are assigned from the most significant bits in each byte to the least significant. For example, a field of two bits followed by a field of six bits has the two bits in the high order bits of the byte.

In the SDL, the notation '=' indicates that in this object, the field read from the bitstream (whose name and type are given in the left hand-side part) is expected to match the value given in the right hand-side part.

The following basic field types are defined. In these definitions, null-terminated means that the last character of a string is Unicode NUL, and hence an empty string is represented by a single Unicode NUL. Some fields using these types may restrict the characters permitted.

Name	Semantics
utf8string	UTF-8 string as defined in IETF RFC 3629, null-terminated.
utfstring	null-terminated string encoded using either UTF-8 or UTF-16. If UTF-16 is used, the sequence of bytes shall start with a byte order mark (BOM) and the null termination shall be 2 bytes set to 0.
utf8list	null-terminated list of space-separated UTF-8 strings
base64string	null-terminated base64 encoded data

4.2.2 Object definitions

An object in this terminology is a box.

Boxes start with a header which gives both size and type. The header permits compact or extended size (32 or 64 bits) and compact or extended types (32 bits or full universal unique identifiers, i.e. UUIDs). The standard boxes all use compact types (32-bit) and most boxes will use the compact (32-bit) size. Typically, only the `MediaDataBox` needs the 64-bit size.

To permit ease of identification, the 32-bit compact type can be expressed as four characters from the range 0020 to 007E, inclusive, of ISO/IEC 10646 (technically identical to the Unicode standard^[28]) or ISO/IEC 8859-1^[34]. Each character is hence expressible in a single byte. The four individual byte values of the field are placed in order in the file. Other fields may also use this 32-bit representation, referred to as a 'four-character code' (4CC). The maintenance of four-character codes used in the format is defined in [Annex D](#).

The size is the entire size of the box, including the size and type header, fields, and all contained boxes. This facilitates general parsing of the file.

User extensions use an extended box type; in this case, the type field is set to 'uuid'.

```
aligned(8) class BoxHeader {
    unsigned int(32) boxtype,
    optional unsigned int(8)[16] extended_type) {
    unsigned int(32) size;
    unsigned int(32) type = boxtype;
    if (size==1) {
        unsigned int(64) largesize;
    } else if (size==0) {
        // box extends to end of file
    }
    if (boxtype=='uuid') {
        unsigned int(8)[16] usertype = extended_type;
    }
}
aligned(8) class Box {
    unsigned int(32) boxtype,
    optional unsigned int(8)[16] extended_type) {
    BoxHeader(boxtype, extended_type);
    // the remaining bytes are the BoxPayload
}
```

The semantics of these two fields are:

size is an integer that specifies the number of bytes in this box, including all its fields and contained boxes; if size is 1 then the actual size is in the field `largesize`; if size is 0, then this box shall be in a top-level container, and be the last box in that container (typically, a file or data object).

delivered over a protocol), and its payload extends to the end of that container (normally only used for a `MediaDataBox`)

`type` identifies the box type; user extensions use an extended type, and in this case, the type field is set to `'uuid'`.

`BoxPayload` is defined as all the bytes in a `Box`, included by the `size` or `largetype` field (as appropriate), following the `BoxHeader`.

Boxes with an unrecognized type shall be ignored and skipped.

Many objects also contain a version number and flags field:

```
aligned(8) class FullBoxHeader(unsigned int(8) v, bit(24) f)
{
    unsigned int(8)    version = v;
    bit(24)           flags = f;
}
aligned(8) class FullBox(unsigned int(32) boxtype,
    unsigned int(8) v, bit(24) f,
    optional unsigned int(8)[16] extended_type)
    extends Box(boxtype, extended_type)
{
    FullBoxHeader(v, f);
    // the remaining bytes are the FullBoxPayload
}
```

The semantics of these two fields are:

`version` is an integer that specifies the version of this format of the box.

`flags` is a map of flags

`FullBoxPayload` is defined as all the bytes in a `FullBox`, included by the `size` or `largetype` field (as appropriate), following the `FullBoxHeader`. Since `BoxPayload` is defined for any box extending from `Box`, `FullBox` has both `BoxPayload` and `FullBoxPayload` defined, the former including the latter (with `BoxPayload` of a `FullBox` including the `FullBoxHeader`).

The payload (contents) of a box are `FullBoxPayload` for a `FullBox` and `BoxPayload` for all boxes.

`FullBoxes` with an unrecognized version shall be ignored and skipped.

NOTE This document and derived specifications describe the different flags defined for a box extending from `FullBox` as hexadecimal values (e.g. `0x000001`); these values correspond to 24-bit unsigned integers with the most significant byte first.

4.2.3 Extensibility of object definitions

The normative objects defined in this document are identified by a 32-bit value, which is normally a four character compact code as defined in 4.2.

To permit user extension of the format, to store new object types, and to permit the inter-operation of the files formatted to this document with certain distributed computing environments, there are a type mapping and a type extension mechanism that together form a pair.

Commonly used in distributed computing are `UUIDs` (universal unique identifiers), which are 16 bytes. Any normative type specified here can be mapped directly into the `UUID` space by composing the four byte type value with the twelve byte ISO reserved value, `0xxxxxxxx-0011-0010-8000-00AA00389B71`. The four character code replaces the `xxxxxxx` in the preceding number. These types are identified to ISO as the object types used in this document.

User objects use the escape type `'uuid'`. They are documented above in subclause 6.3. After the size and type fields, there is a full 16-byte `UUID`.

Systems which wish to treat every object as having a `UUID` could employ the following algorithm:

```
size := read_uint32();
type := read_uint32();
if (type=='uuid')
    then uuid := read_uuid()
    else uuid := form_uuid(type, ISO_12_bytes);
```

Similarly when linearizing a set of objects into files formatted to this document, the following is applied:

```
write_uint32( object_size(object) );
uuid := object_uuid_type(object);
if (is_ISO_uuid(uuid) )
    write_uint32( ISO_type_of(uuid) )
    else { write_uint32('uuid'); write_uuid(uuid); }
```

A file containing boxes from this document that have been written using the 'uuid' escape and the full UUID is not compliant; systems are not required to recognize standard boxes written using the 'uuid' and an ISO UUID.

4.3 File-type box

4.3.1 Definition

Box Type: 'ftyp'
Container: File, or OriginalFileTypeBox
Mandatory: Yes
Quantity: Exactly one (but see below)

Files conformant to [Clause 4](#) shall contain a FileTypeBox. For compatibility with an earlier edition of this document, files may be conformant to this document and not contain a FileTypeBox. Files with no FileTypeBox should be read as if they contained a FileTypeBox with Major_brand='mp41', minor_version=0, and the single compatible brand 'mp41'.

A media-file structured to this document may be compatible with more than one detailed specification, and it is therefore not always possible to speak of a single 'type' or 'brand' for the file. This means that the utility of the file name extension and multipurpose internet mail extension (MIME) type are somewhat reduced.

This box shall be placed as early as possible in the file (e.g. after any obligatory signature, but before any significant variable-size boxes such as a MovieBox, MediaDataBox, or FreeSpaceBox). It identifies which specification is the 'best use' of the file, and a minor version of that specification; and also a set of other specifications to which the file complies. Readers implementing this format should attempt to read files that are marked as compatible with any of the specifications that the reader implements. Any incompatible change in a specification should therefore register a new 'brand' identifier to identify files conformant to the new specification.

The minor version is informative only. It does not appear for compatible-brands, and is not used to determine the conformance of a file to a standard. It may allow more precise identification of the major specification, for inspection, debugging, or improved decoding.

Files would normally be externally identified (e.g. with a file extension or mime type) that identifies the 'best use' (major brand), or the brand that the author believes will provide the greatest compatibility.

This subclause does not define any brands. However, see [subclause 6.3](#) for brands for files conformant to the whole document and not just this subclause. All file format brands defined in this document are included in [Annex E](#) with a summary of which features they require.

4.3.2 Syntax

```
aligned(8) class GeneralTypeBox(code) extends Box(code) {
    unsigned int(32)    major_brand;
    unsigned int(32)    minor_version;
    unsigned int(32)    compatible_brands[];    // to end of the box
}
```

```
aligned(8) class FileTypeBox extends GeneralTypeBox ('ftyp')
{ }
```

4.3.3 Semantics

This box identifies the specifications to which this file complies.

Each brand is a four character code, registered with ISO, that identifies a precise specification.

`major_brand` – is a brand identifier

`minor_version` – is an informative integer for the minor version of the major brand

`compatible_brands` – is a list, to the end of the box, of brands

4.4 Extended type box

4.4.1 Definition

Box Type: 'etyp'

Container: File, ItemPropertyContainerBox, or OriginalFileTypeBox

Mandatory: No

Quantity: Zero or more in file, zero or one per an item

Box Type: 'tyco'

Container: ExtendedTypeBox

Mandatory: Yes

Quantity: One or more

The `ExtendedTypeBox` may be placed after the `FileTypeBox`, any `SegmentTypeBox`, or any `TrackTypeBox`, or used as an item property to indicate that a reader should only process the file, the segment, the track, or the item, respectively, if it supports the processing requirements of all the brands in at least one of the contained `TypeCombinationBoxes`, or at least one brand in the preceding `FileTypeBox`, `SegmentTypeBox`, or `TrackTypeBox`, or in the `BrandProperty` associated with the same item, respectively.

The `TypeCombinationBox` expresses that the associated file, segment, track, or item may contain any boxes or other code points required to be supported in any of the brands listed in the `TypeCombinationBox` and that the associated file, segment, track, or item complies with the intersection of the constraints of the brands listed in the `TypeCombinationBox`.

NOTE Effectively, if the `compatible_brands` in the `FileTypeBox` are labeled `C[1]`, `C[2]`, etc., and the `compatible_brands` in the first `TypeCombinationBox` `D[1,1]`, `D[1,2]`, and the second `D[2,1]`, `D[2,2]`, and so on, then a reader has to support:

```
C[1]
or C[2]
or ...
or ( D[1,1] and D[1,2] and ...)
or ( D[2,1] and D[2,2] and ...)
...
```

4.4.2 Syntax

```
aligned(8) class TypeCombinationBox extends Box('tyco') {
    unsigned int(32) compatible_brands[]; // to end of the box
}
aligned(8) class ExtendedTypeBox extends Box('etyp') {
    TypeCombinationBox compatible_combinations[]; // to end of the box
}
```

4.4.3 Semantics

`compatible_brands` is a list of brands.

`compatible_combinations` is a list of `TypeCombinationBoxes`.

5 Structure of this document

[Clause 4](#) defines object-structured files; files that are built from boxes.

[Clause 6](#) gives the core concepts and data-types for time-based presentations, called 'movies' in this document.

[Clause 8](#) defines the boxes used by time-based presentations, and other formats.

[Clause 9](#) defines the hint track formats used to support some streaming protocols.

[Clause 10](#) builds on the concept of sample groups as defined in [8.9](#) and defines some sample groups.

[Clause 11](#) defines how to base a file format on this specification.

[Clause 12](#) builds on the general concepts of tracks as defined in Clause 8, and defines track formats for various general types of media (video, sound, etc.).

[Annex A](#) provides an informative introduction to time-based presentations, which may be of assistance to first-time readers and implementers.

[Annex B](#) provides guidance on writing derived specifications.

[Annex C](#) provides the syntax for uniform resource identifier (URI) fragments.

[Annex D](#) documents how identifier values defined externally to this document are managed.

[Annex E](#) defines brands that may be used to identify the conformance and reader requirements to the structures defined in this document for time-based presentations.

[Annex F](#) contains the formal IANA registration of segments.

[Annex G](#) defines some forms used for labelling metadata with uniform resource identifier (URI) labels.

[Annex H](#) provides an overview of the use of hint tracks for RTP streams and RTP stream reception.

[Annex I](#) contains the formal definitions of the types of stream access points in timed media streams.

[Annex J](#) contains examples of the use of the `SegmentIndexBox` defined in [8.16.3](#).

[Annex K](#) defines the MIME parameters that may be used to annotate MIME types for time-based presentations based on [Clause 6](#).

6 ISO base media file organization

6.1 Files, segments, and streams

This document supports the exchange of presentations in three principle ways:

- 1) As a single file (e.g. on exchangeable media such as discs, or as a download).
- 2) As a series of segments, preceded by an initialization segment.
- 3) Transformed by supporting structures, called hint tracks, into a streaming protocol such as the IETF real-time protocol RTP [\[\[ref\]\]](#) or an MPEG-2 transports stream [\[\[ref\]\]](#).

A presentation file logically includes all its segments.

The file format supports transformation of media data into a streaming protocol as well as local playback. The process of sending protocol data units is time-based, just like the display of time-based data, and is therefore suitably described by a time-based format. A file or 'movie' that supports streaming includes information about the data units to stream. This information is included in additional tracks of the file

called “hint” tracks. Hint tracks may also be used to record a stream; these are called Reception Hint Tracks, to differentiate them from plain (or server, or transmission) hint tracks.

6.2 Presentation structure

6.2.1 Object structure of a presentation

The presentation file is an object-structured file as defined in [Clause 4](#) (and hence contains a `FileTypeBox`); some of these objects may contain other objects.

Presentation files contain data that structures, orders, times, and describes the media data that is passed to decoders. This non-media data is called structure-data here.

The sequence of objects in a presentation shall contain exactly one `MovieBox`. It is usually close to the beginning or end of the file, to permit its easy location. The other objects found at this level include a `FileTypeBox`, `FreeSpaceBox(es)`, `MovieFragmentBox(es)`, a `MetaBox`, or `MediaDataBox(es)`.

6.2.2 Meta data and media data

The structure-data is contained within a structure-data wrapper (the `MovieBox` or `MovieFragmentBox`); the media data is contained either in the same file, within `MediaDataBox(es)`, or in other files. The media data is composed, for example, of images or audio data; the media data containers, or media data files, may contain other un-referenced information.

The file containing the presentation structure-data may also contain all the media data, whereupon the presentation is self-contained. If the media data is externally referenced in other files, they are not required to be formatted to this document; they are used to contain media data, and may also contain unused media data, or other information. This document concerns the structure of the presentation file only. The format of the media-data files is constrained by this document only in that the media-data in the media files needs to be capable of description by the structure-data defined here.

These other files may be ISO files, image files, or other formats. Only the media data itself, such as JPEG 2000 images, is stored in these other files; all timing and framing (position and size) information is in the ISO base media file, so the ancillary files are essentially free-format.

6.3 Structure-data (objects)

6.3.1 Box

Type fields not defined here are reserved. Private extensions shall be achieved through the 'uuid' type. In addition, the following types are not and will not be used, or used only in their existing sense, in future editions of this document, to avoid conflict with existing content using earlier pre-standard versions of this format:

`clip`, `crgn`, `matt`, `kmat`, `pnot`, `ctab`, `load`, `imap`;

these track reference types (as found in the `reference_type` of a `TrackReferenceBox`):

`tmcd`, `chap`, `sync`, `scpt`, `ssrc`.

A number of boxes contain index values into sequences in other boxes. These indexes start with the value 1 (1 is the first entry in the sequence).

6.3.2 Data types and fields

In a number of boxes in this document, there are two variant forms: version 0 using 32-bit fields, and version 1 using 64-bit sizes for those same fields. In general, if a version 0 box (32-bit field sizes) can be used, it should be; version 1 boxes should be used only when the 64-bit field sizes they permit, are required. Values for counters, offsets, times, durations etc. in this format do not ‘wrap’ to 0 when the

maximum value that can be stored in their field is reached; appropriately large fields must be used for all values.

For convenience during content creation there are creation and modification times stored in the file. These can be 32-bit or 64-bit numbers, counting seconds since midnight, Jan. 1, 1904, which is a convenient date for leap-year calculations. 32 bits are sufficient until approximately year 2040. These times shall be expressed in Universal Time Coordinated (UTC) as defined in ITU-R TF.460-6:2002, Annex I, and therefore may need adjustment to local time if displayed.

Fixed-point numbers are signed or unsigned values resulting from dividing an integer by an appropriate power of 2. For example, a 30.2 fixed-point number is formed by dividing a 32-bit integer by 4.

Fields shown as “template” in the box descriptions are fields which are coded with a default value unless a derived specification defines their use and permits writers to use other values than the default. If the field is used in another specification, that use must be conformant with its definition here, and the specification must define whether the use is optional or mandatory. Similarly, fields marked “pre-defined” were used in an earlier edition of this document. For both kinds of fields, if a field of that kind is not used in a specification, then it should be set to the indicated default value. If the field is not used it shall be copied un-inspected when boxes are copied, and ignored on reading.

Matrix values which occur in the headers specify a transformation of video images for presentation. Not all derived specifications use matrices; if they are not used, they shall be set to the identity matrix. If a matrix is used, the point (p,q) is transformed into (p', q') using the matrix as follows:

$$\begin{pmatrix} p & q & 1 \end{pmatrix} * \begin{vmatrix} a & b & u \\ c & d & v \\ x & y & w \end{vmatrix} = \begin{pmatrix} m & n & z \end{pmatrix}$$

$$m = ap + cq + x; \quad n = bp + dq + y; \quad z = up + vq + w;$$

$$p' = m/z; \quad q' = n/z$$

The coordinates {p,q} are on the decompressed frame, and {p', q'} are at the rendering output. Therefore, for example, the matrix {2,0,0, 0,2,0, 0,0,1} exactly doubles the pixel dimension of an image. The coordinates transformed by the matrix are not normalized in any way, and represent actual sample locations. Therefore {x,y} can, for example, be considered a translation vector for the image.

The coordinate origin is located at the upper left corner, and X values increase to the right, and Y values increase downwards. {p,q} and {p',q'} are to be taken as absolute pixel locations relative to the upper left hand corner of the original image (after scaling to the size determined by the track header's width and height) and the transformed (rendering) surface, respectively.

Each track is composed using its matrix as specified into an overall image; this is then transformed and composed according to the matrix at the movie level in the `MovieHeaderBox`. It is application-dependent whether the resulting image is ‘clipped’ to eliminate pixels, which have no display, to a vertical rectangular region within a window, for example. So for example, if only one video track is displayed and it has a translation to {20,30}, and a unity matrix is in the `MovieHeaderBox`, an application may choose not to display the empty “L” shaped region between the image and the origin.

All the values in a matrix are stored as 16.16 fixed-point values, except for u, v and w, which are stored as 2.30 fixed-point values.

The values in the matrix are stored in the order {a,b,u, c,d,v, x,y,w}.

6.3.3 URIs as type indicators

When URIs are used as a type indicator (e.g. in a sample entry or for un-timed metadata), the URI shall be absolute, not relative and the format and meaning of the data must be defined by the URI in question. This identification may be hierarchical, in that an initial sub-string of the URI might identify the overall nature or family of the data (e.g. urn:oid: identifies that the metadata is labelled by an ISO-standard object identifier).

The URI should be, but is not required to be, de-referencable. It may be string compared by readers with the set of URI types it knows and recognizes. URIs provide a large non-colliding non-registered space for type identifiers.

If the URI contains a domain name (e.g. it is a URL), then it should also contain a month-date in the form mmyyyy. That date shall be near the time of the definition of the extension, and it must be true that the URI was defined in a way authorized by the owner of the domain name at that date. (This avoids problems when domain names change ownership).

6.3.4 Box order

An overall view of the normal encapsulation structure is provided in the following informative [Table 1](#). In the event of a conflict between this table and the prose, the prose prevails. The order of boxes within its container is not necessarily indicated in [Table 1](#).

The table shows those boxes that may occur at the top-level in the left-most column; indentation is used to show possible containment. Thus, for example, a `TrackHeaderBox` (`tkhd`) is found in a `TrackBox` (`trak`), which is found in a `MovieBox` (`moov`). Not all boxes need to be used in all files; the mandatory boxes are marked with an asterisk (*). See the description of the individual boxes for a discussion of what must be assumed if the optional boxes are not present.

Objects using an extended type may be placed in a wide variety of containers, not just the top level.

In order to improve interoperability and utility of the files, the following rules and guidelines shall be followed for the order of boxes:

- 1) The `FileTypeBox` shall occur before any variable-length box (e.g. movie, free space, media data). Only a fixed-size box such as a file signature, if required, may precede it.
- 2) It is strongly **recommended** that all header boxes be placed first in their container: these boxes are the `MovieHeaderBox`, `TrackHeaderBox`, `MediaHeaderBox`, and the specific media headers inside the `MediaInformationBox` (e.g. the `VideoMediaHeaderBox`).
- 3) Any movie fragment boxes **should** be in sequence order (see subclause [8.8.5](#)).
- 4) It is **recommended** that the boxes within the `SampleTableBox` be in the following order: `SampleDescriptionBox`, `TimeToSampleBox`, `SampleToChunkBox`, `SampleSizeBox`, `ChunkOffsetBox`.
- 5) It is strongly **recommended** that the `TrackReferenceBox` and `EditBox` (if any) **should** precede the `MediaBox`, and the `HandlerBox` **should** precede the `MediaInformationBox`, and the `DataInformationBox` **should** precede the `SampleTableBox`.
- 6) It is **recommended** that `UserDataBoxes` be placed last in their container.
- 7) It is **recommended** that the `MovieFragmentRandomAccessBox`, if present, be last in the file.
- 8) It is **recommended** that the `ProgressiveDownloadInfoBox` be placed as early as possible in files, for maximum utility.

Table 1 — Box types, structure and cross-reference

Box types, structure, and cross-reference							
ftyp					*	4.2.3	<i>file type and compatibility</i>
otyp						8.19.5	<i>original file-type</i>
pdin						8.1.3	<i>progressive download information</i>
moov					*	8.2.1	<i>container for all the structure-data</i>
	mvhd				*	8.2.2	<i>movie header, overall declarations</i>
	meta					8.11.1	<i>metadata</i>
	trak				*	8.3.1	<i>container for an individual track or stream</i>

Table 1 (continued)

Box types, structure, and cross-reference								
		tkhd			*	8.3.2	track header, overall information about the track	
		tref				8.3.3	track reference container	
		trgr				8.3.4	track grouping indication	
		edts				8.6.5	edit list container	
			elst			8.6.6	an edit list	
		meta				8.11.1	metadata	
		mdia			*	8.4	container for the media information in a track	
			mdhd		*	8.4.2	media header, overall information about the media	
			hdlr		*	8.4.3	handler, declares the media (handler) type	
			elng			8.4.6	extended language tag	
			minf		*	8.4.4	media information container	
				vmhd		12.1.2	video media header, overall information (video track only)	
				smhd		12.2.2	sound media header, overall information (sound track only)	
				hmhd		12.4.3	hint media header, overall information (hint track only)	
				sthd		12.6.2	subtitle media header, overall information (subtitle track only)	
				nmhd		8.4.5.2	Null media header, overall information (some tracks only)	
				dinf	*	8.7.1	data information box, container	
					dref	*	8.7.2 data reference box, declares source(s) of media data in track	
				stbl	*	8.5.1	sample table box, container for the time/space map	
				stsd	*	8.5.2	sample descriptions (codec types, initialization etc.)	
				stts	*	8.6.1.2	(decoding) time-to-sample	
				ctts		8.6.1.3	(composition) time to sample	
				cslg		8.6.1.4	composition to decode timeline mapping	
				stsc	*	8.7.4	sample-to-chunk, partial data-offset information	
				stsz		8.7.3.2	sample sizes (framing)	
				stz2		8.7.3.3	compact sample sizes (framing)	
				stco	*	8.7.5	chunk offset, partial data-offset information	
				co64		8.7.5	64-bit chunk offset	
				stss		8.6.2	sync sample table	
				stsh		8.6.3	shadow sync sample table	
				padb		8.7.6	sample padding bits	
				stdp		8.7.6	sample degradation priority	
				sdtg		8.6.4	independent and disposable samples	
				sbgp		8.9.2	sample-to-group	
				sgpd		8.9.3	sample group description	
				subs		8.7.7	sub-sample information	
				saiz		8.7.8	sample auxiliary information sizes	
				saio		8.7.9	sample auxiliary information offsets	
		udta				8.10.1	user-data	
			cpri			8.10.2	copyright etc.	
			tsel			8.10.3	track selection box	
			kind			8.10.4	track kind box	

Table 1 (continued)

Box types, structure, and cross-reference							
			strk			8.14.3	sub track box
				stri		8.14.4	sub track information box
				strd		8.14.5	sub track definition box
			ludt			12.2.7	audio stream loudness
	mvex					8.8.1	movie extends box
		mehd				8.8.2	movie extends header box
		trex			*	8.8.3	track extends defaults
		leva				8.8.13	level assignment
	udta					8.10.1	user-data
		cpri				8.10.2	copyright etc.
moof						8.8.4	movie fragment
	mfhd				*	8.8.5	movie fragment header
	meta					8.11.1	metadata
	traf					8.8.6	track fragment
		tfhd			*	8.8.7	track fragment header
		trun				8.8.8	track fragment run
		sbgp				8.9.2	sample-to-group
		sgpd				8.9.3	sample group description
		subs				8.7.7	sub-sample information
		saiz				8.7.8	sample auxiliary information sizes
		saio				8.7.9	sample auxiliary information offsets
		tfdt				8.8.12	track fragment decode time
		meta				8.11.1	metadata
		udta				8.10.1	user-data
	udta					8.10.1	user-data
mfra						8.8.9	movie fragment random access
	tfra					8.8.10	track fragment random access
	mfro				*	8.8.11	movie fragment random access offset
mdat						8.2.2	media data container
free						8.1.2	free space
skip						8.1.2	free space
imda						8.1.4	media data container that contains an identifier to be used with data references
meta						8.11.1	metadata
	hdlr				*	8.4.3	handler, declares the metadata (handler) type
	dinf					8.7.1	data information box, container
		dref				8.7.2	data reference box, declares source(s) of metadata items
	iloc					8.11.2.3	item location
	ipro					8.11.5	item protection
		sinf				8.12.2	protection scheme information box
			frma			8.12.3	original format box
			schm			8.12.6	scheme type box
			schl			8.12.7	scheme information box
	iinf					8.11.6	item information

Table 1 (continued)

Box types, structure, and cross-reference							
	xml					8.11.2	XML container
	bxm1					8.11.2	binary XML container
	pitm					8.11.4	primary item reference
	fiin					8.13.2	file delivery item information
		paen				8.13.2	partition entry
			fire			8.13.7	file reservoir
			fpar			8.13.3	file partition
			fecr			8.13.4	FEC reservoir
		segr				8.13.5	file delivery session group
		gitn				8.13.6	group id to name
	idat					8.11.11	item data
	iref					8.11.12	item reference
styp						8.16.2	segment type
sidx						8.16.3	segment index
ssix						8.16.4	subsegment index
prft						8.16.5	producer reference time
!mov						8.19.6	compressed movie box
!mof						8.19.7	compressed movie fragment box
!six						8.19.8	compressed segment index box
!ssx						8.19.9	compressed subsegment index box

6.4 Time structure overview

Tracks identify, in *decoding order*, of a sequence of *samples*. Each sample has a *decoding timestamp* that is computed by adding to the previous sample's decoding timestamp, the previous sample's duration (as given by the values in the `TimeToSampleBox` or the equivalent field in movie fragments). The decoding timestamp of the first sample is defined as being at time zero. This forms the decoding timeline of a track.

In some coding systems (notably video) samples are coded in a different order from their presentation order. In this case, each sample is assigned a *composition time* (as given by the values in the `CompositionOffsetBox` or the equivalent field in movie fragments), which is computed by adding a *composition offset* to the decoding timestamp. This forms the composition timeline, and re-orders the samples into *composition order*.

The *presentation timeline* for each track is formed by the concatenation of sections of the composition timeline by means of explicit or implicit *edit lists*.

The presentation timelines of all the tracks are aligned at their zero point. This forms the presentation timeline for the presentation as a whole. This aligned zero point is the nominal time at which presentation starts and from which movie and track durations are measured.

Composition offsets can be signed or unsigned. When they are unsigned, the composition timeline and the decoding timeline are related, and the decoding and compositions times can be considered as enabling a hypothetical system, that is capable of instant decoding, to manage buffering. When they are signed, the composition timeline and the decoding timeline are disconnected. They can be reconnected using the `CompositionToDecodeBox`, if needed, though many systems do not use decoding timestamps, merely decoding in decoding order.

6.5 Identifiers

The track identifiers used in an ISO file are unique within that file; no two tracks shall use the same identifier. Under unified identifier handling (see [E.18](#)) this uniqueness requirement is extended to other identifiers.

The next track identifier value stored in `next_track_ID` in the `MovieHeaderBox` generally contains a value one greater than the largest identifier value, of the set required to be unique, found in the file. This enables easy generation of a unique identifier under most circumstances. However, if this value is equal to ones (32-bit unsigned maxint), then a search for an unused unique identifier is needed for all additions.

6.6 Brand identification

The definitions of the brands that apply to timed presentations are specified in [Annex E](#).

6.7 Uniform resource locators (URLs)

When a file conformant to this document is identified by URL, a URL fragment (if present) shall follow the syntax(es) documented in [Annex C](#).

When files are identified by a MIME type, the syntax specified in IETF RFC 6381^[29] and [Annex K](#) shall be used for any parameters.

7 Streaming support

Segmented streaming is supported by segments as defined in [3.1.41](#) and [8.16](#).

Transformed streaming into streaming protocols is supported by hint tracks as defined in [12.4](#), with specific formats for some protocols in [Clause 9](#); there is a general overview in [A.12](#).

8 Box structures

8.1 File structure and general boxes

8.1.1 Media data box

8.1.1.1 Definition

Box Type: 'mdat'

Container: File

Mandatory: No

Quantity: Zero or more

This box contains the media data. In video tracks, this box would contain video frames. A presentation may contain zero or more `MediaDataBoxes`. The actual media data follows the type field; its structure is described by the structure-data (see particularly the `SampleTableBox`, subclause [8.5](#), and the `ItemLocationBox`, subclause [8.11.2.3](#)).

In large presentations, it may be desirable to have more data in this box than a 32-bit size would permit. In this case, the large variant of the size field, above in subclause [4.2](#), is used.

There may be any number of these boxes in the file (including zero, if all the media data is in other files). The structure-data refers to media data by its absolute offset within the file (see subclause [8.7.5](#), the `ChunkOffsetBox`); so `MediaDataBox` headers and free space may easily be skipped, and files without any box structure may also be referenced and used.

8.1.1.2 Syntax

```
aligned(8) class MediaDataBox extends Box('mdat') {
    bit(8) data[];
}
```

8.1.1.3 Semantics

`data` is the contained media data

8.1.2 Free space box

8.1.2.1 Definition

Box Types: 'free', 'skip'
Container: File or other box
Mandatory: No
Quantity: Zero or more

The contents of a `FreeSpaceBox` are irrelevant and may be ignored, or the box deleted, without affecting the presentation. (Care should be exercised when deleting the box, as this may invalidate offsets used to refer to other data, unless this box is after all the media data).

8.1.2.2 Syntax

```
aligned(8) class FreeSpaceBox extends Box('free_type') {
    unsigned int(8) data[];
}
```

8.1.2.3 Semantics

`free_type` shall be 'free' or 'skip'.

8.1.3 Progressive download information box

8.1.3.1 Definition

Box Types: 'pdin'
Container: File
Mandatory: No
Quantity: Zero or One

The `ProgressiveDownloadInfoBox` aids the progressive download of an ISO file. The box contains pairs of numbers (to the end of the box) specifying combinations of effective file download bitrate in units of bytes/sec and a suggested initial playback delay in units of milliseconds.

A receiving party can estimate the download rate it is experiencing, and from that obtain an upper estimate for a suitable initial delay by linear interpolation between pairs, or by extrapolation from the first or last entry.

It is recommended that the `ProgressiveDownloadInfoBox` be placed as early as possible in files, for maximum utility.

8.1.3.2 Syntax

```
aligned(8) class ProgressiveDownloadInfoBox
    extends FullBox('pdin', version = 0, 0) {
    for (i=0; ; i++) { // to end of box
        unsigned int(32) rate;
        unsigned int(32) initial_delay;
    }
}
```

8.1.3.3 Semantics

`rate` is a download rate expressed in bytes/second

`initial_delay` is the suggested delay to use when playing the file, such that if download continues at the given rate, all data within the file will arrive in time for its use and playback should not need to stall.

8.1.4 Identified media data box

8.1.4.1 Definition

Box Type: 'imda'

Container: File

Mandatory: No

Quantity: Zero or more

This box contains the media data. Its semantics are the same as those for `MediaDataBox` but it additionally contains an identifier that is used in setting up data references to the contained media data.

8.1.4.2 Syntax

```
aligned(8) class IdentifiedMediaDataBox extends Box('imda') {
    unsigned int(32) imda_identifier;
    bit(8) data[]; // until the end of the box
}
```

8.1.4.3 Semantics

`imda_identifier` shall differ from the `imda_identifier` values of the other

`IdentifiedMediaDataBoxes` of the file.

8.2 Movie structure

8.2.1 Movie box

8.2.1.1 Definition

Box Type: 'moov'

Container: File

Mandatory: Yes

Quantity: Exactly one

The structure-data for a presentation is stored in the single `MovieBox` which occurs at the top-level of a file. Normally this box is close to the beginning or end of the file, though this is not required.

8.2.1.2 Syntax

```
aligned(8) class MovieBox extends Box('moov') {
}
```

8.2.2 Movie header box

8.2.2.1 Definition

Box Type: 'mvhd'

Container: `MovieBox`

Mandatory: Yes

Quantity: Exactly one

This box defines overall information which is media-independent, and relevant to the entire presentation considered as a whole.

The duration of a movie is usually the duration of the longest track (as documented by the `duration` field of the `TrackHeaderBox`); if any track has an indefinite duration, the movie should also have an indefinite duration. If there are tracks longer than the movie duration, it is recommended that playback terminate at the movie duration (i.e. that the longer tracks be implicitly trimmed to the movie duration); this can happen when two media have frame durations that are not exactly divisible, for example.

8.2.2.2 Syntax

```
aligned(8) class MovieHeaderBox extends FullBox('mvhd', version, 0) {
    if (version==1) {
        unsigned int(64)    creation_time;
        unsigned int(64)    modification_time;
        unsigned int(32)    timescale;
        unsigned int(64)    duration;
    } else { // version==0
        unsigned int(32)    creation_time;
        unsigned int(32)    modification_time;
        unsigned int(32)    timescale;
        unsigned int(32)    duration;
    }
    template int(32)    rate = 0x00010000; // typically 1.0
    template int(16)    volume = 0x0100; // typically, full volume
    const bit(16)    reserved = 0;
    const unsigned int(32)[2]    reserved = 0;
    template int(32)[9]    matrix =
        { 0x00010000,0,0,0,0x00010000,0,0,0,0x40000000 };
    // Unity matrix
    bit(32)[6]    pre_defined = 0;
    unsigned int(32)    next_track_ID;
}
```

8.2.2.3 Semantics

`version` is an integer that specifies the version of this box (0 or 1 in this document)

`creation_time` is an integer that declares the creation time of the presentation (in seconds since midnight, Jan. 1, 1904, in UTC time)

`modification_time` is an integer that declares the most recent time the presentation was modified (in seconds since midnight, Jan. 1, 1904, in UTC time)

`timescale` is an integer that specifies the time-scale for the entire presentation; this is the number of time units that pass in one second. For example, a time coordinate system that measures time in sixtieths of a second has a time scale of 60.

`duration` is an integer that declares length of the presentation (in the indicated timescale). This property is derived from the presentation's tracks: the value of this field corresponds to the duration of the longest track in the presentation. If the duration cannot be determined then duration is set to all 1s.

`rate` is a fixed point 16.16 number that indicates the preferred rate to play the presentation; 1.0 (0x00010000) is normal forward playback

`volume` is a fixed point 8.8 number that indicates the preferred playback volume. 1.0 (0x0100) is full volume.

`matrix` provides a transformation matrix for the video; (u,v,w) are restricted here to (0,0,1), hex values (0,0,0x40000000).

`next_track_ID` is a non-zero integer that indicates a value to use for the `track_ID` of the next track to be added to this presentation. Zero is not a valid `track_ID` value. The value of `next_track_ID` shall be larger than the largest `track_ID` in use. If this value is equal to all 1s (32-bit maxint), and a new media track is to be added, then a search must be made in the file for an unused value of `track_ID`.

8.3 Track structure

8.3.1 Track box

8.3.1.1 Definition

Box Type: 'trak'
 Container: `MovieBox`
 Mandatory: Yes
 Quantity: One or more

This is a container box for a single track of a presentation. A presentation consists of one or more tracks. Each track carries its own temporal and spatial information. Each track will contain its associated `MediaBox`.

Tracks are used for a number of purposes, including: (a) to contain media data (media tracks) and (b) to contain packetization information for streaming protocols (hint tracks).

There shall be at least one media track within a `MovieBox`, and all the media tracks that contributed to the hint tracks shall remain in the file, even if the media data within them is not referenced by the hint tracks; after deleting all hint tracks, the entire un-hinted presentation shall remain.

8.3.1.2 Syntax

```
aligned(8) class TrackBox extends Box('trak') {
}
```

8.3.2 Track header box

8.3.2.1 Definition

Box Type: 'tkhd'
 Container: `TrackBox`
 Mandatory: Yes
 Quantity: Exactly one

This box specifies the characteristics of a single track. Exactly one `TrackHeaderBox` is contained in a track.

In the absence of an edit list, the presentation of a track starts at the beginning of the overall presentation. An 'empty' edit is used to offset the start time of a track (see 8.6.6).

The tracks marked with the `track_in_movie` flag set to 1 are those that are intended by the file writer for direct presentation. Thus a track that is used as input to another track — either before or after decoding — but that is not presented by itself — should have the `track_in_movie` flag set to 0. Tracks having the `track_in_movie` flag set are candidates for playback, regardless of whether they are media tracks or reception hint tracks. A track may be used as input and also have the `track_in_movie` flag set to 1.

If a track is a member of a group that presents alternatives for presentation, either by using the `alternate_group` field in the track header, or by using the `TrackGroupBox` with a group type that defines alternatives, then either only the preferred or default choice track should have the `track_in_movie` flag set to 1, or all tracks should have the `track_in_movie` flag set to 1 (if no default is to be indicated).

If an 'altr' entity group contains entities for playing, `track_in_movie` should be equal to 1 in all the tracks of the entity group.

If in a presentation no tracks have `track_in_movie` set, and therefore it appears that there is nothing to present, then a player may enable a track from each group for presentation; derived specifications can give further guidance and/or restrictions.

Tracks that are marked as not enabled (`track_enabled` set to 0) shall be ignored and treated as if not present. Application environments may offer a way to enable/disable tracks at run-time and dynamically alter the state of this flag.

The processing of the pixel data from the output of the decoder to its rendering on the screen is not a conformance point of ISO/BMFF. However, several structures enable signaling of such rendering features. They are based on the following assumptions:

- 1) Under the 'iso3' brand or brands that share its requirements, the `TrackHeaderBox` `width` and `height` assume that the rendered pixels are square (i.e. that the pixel aspect ratio is 1:1). For other brands, the use of the associated structure for rendering is undetermined.
- 2) The `VisualSampleEntry` documents the expected size of the pixel buffer needed to receive the codec output, possibly cropped by in-stream structures.

EXAMPLE If a video codec works only with multiples of 16 pixels per line or column, and if the width and height of the video fed to the encoder is 1000x500, the encoder will typically, internally use 64x32 blocs of pixels, but it is expected to use codec-specific cropping structures, that are not exposed at the ISO/BMFF level to output a 1000x500 video. The `width` and `height` of the `VisualSampleEntry` fields in this case will be 1000x500.

- 3) The `PixelAspectRatioBox` documents the aspect ratio that should be applied to the pixels output by the decoder but does not imply the adjustment. The associated adjustment should be taken care of by setting scaled values in the `TrackHeaderBox`.

EXAMPLE If prior to the encoder, the pixels are scaled horizontally by a factor of 1/2, at the output of the decoder, the inverse scaling should be applied to present non distorted videos. This is done by setting the `TrackHeaderBox` `height` equal to the `VisualSampleEntry` `height` and the `TrackHeaderBox` `width` equal to the double of the `VisualSampleEntry` `width`. Additionally, a `PixelAspectRatioBox` with a `hSpacing` twice bigger than its `vSpacing` field may be used, if in-stream structures do not carry that information.

- 4) Additionally, a `CleanApertureBox` may be provided to further crop the video.

The processing of the decoded pixel is assumed to be as follows:

- 1) Any cropping documented by a `CleanApertureBox` is applied on the pixels output by the decoder,
- 2) Then, if a `CleanApertureBox` is present, the cropped image is then scaled horizontally by the factor $\text{TrackHeaderBox.width} / \text{CleanAperture.width}$ and vertically by $\text{TrackHeaderBox.height} / \text{CleanAperture.height}$
- 3) Otherwise, if a `CleanApertureBox` is not present, the decoded image is then scaled horizontally by the factor $\text{TrackHeaderBox.width} / \text{SampleEntry.width}$ and vertically by $\text{TrackHeaderBox.height} / \text{SampleEntry.height}$.

NOTE 1 This operation is called in previous editions "normalization to track dimensions".

- 4) The `TrackHeaderBox` matrix is then applied
- 5) All visual tracks are superposed in increasing order of the `TrackHeaderBox.layer` value
- 6) The `MovieHeaderBox` matrix is then applied to the composition.

NOTE 2 This is a theoretical processing model and concrete implementations following it should avoid resampling the image, in particular when the combination of the above operations results in the identity transformation.

The duration field here does not include the duration of following movie fragments, if any, but only of the media in the enclosing `MovieBox`. The `MovieExtendsHeaderBox` may be used to document the duration including movie fragments, when desired and possible.

8.3.2.2 Syntax

```
aligned(8) class TrackHeaderBox
  extends FullBox('tkhd', version, flags){
    if (version==1) {
      unsigned int(64)    creation_time;
      unsigned int(64)    modification_time;
      unsigned int(32)    track_ID;
      const unsigned int(32) reserved = 0;
      unsigned int(64)    duration;
    } else { // version==0
      unsigned int(32)    creation_time;
      unsigned int(32)    modification_time;
      unsigned int(32)    track_ID;
      const unsigned int(32) reserved = 0;
      unsigned int(32)    duration;
    }
    const unsigned int(32)[2] reserved = 0;
    template int(16) layer = 0;
    template int(16) alternate_group = 0;
    template int(16) volume = {if track_is_audio 0x0100 else 0};
    const unsigned int(16) reserved = 0;
    template int(32)[9] matrix=
      { 0x00010000,0,0,0,0x00010000,0,0,0,0x40000000 };
      // unity matrix
    unsigned int(32) width;
    unsigned int(32) height;
  }
```

8.3.2.3 Semantics

`version` is an integer that specifies the version of this box (0 or 1 in this document)

`flags` is a 24-bit integer with flags; the following values are defined:

`track_enabled`: Flag mask is 0x000001. The value 1 indicates that the track is enabled. A disabled track (when the value of this flag is zero) is treated as if it were not present.

`track_in_movie`: Flag mask is 0x000002. The value 1 indicates that the track, or one of its alternatives (if any) forms a direct part of the presentation. The value 0 indicates that the track does not represent a direct part of the presentation.

`track_in_preview`: Flag mask is 0x000004. This flag currently has no assigned meaning, and the value should be ignored by readers. In the absence of further guidance (e.g. from derived specifications), the same value as for `track_in_movie` should be written.

`track_size_is_aspect_ratio`: Flag value is 0x000008. The value 1 indicates that the width and height fields are not expressed in pixel units. The values have the same units but these units are not specified. The values are only an indication of the desired aspect ratio. If the aspect ratios of this track and other related tracks are not identical, then the respective positioning of the tracks is undefined, possibly defined by external contexts.

`creation_time` is an integer that declares the creation time of this track (in seconds since midnight, Jan. 1, 1904, in UTC time).

`modification_time` is an integer that declares the most recent time the track was modified (in seconds since midnight, Jan. 1, 1904, in UTC time).

`track_ID` is an integer that uniquely identifies this track over the entire life-time of this presentation; `track_IDs` are never re-used and cannot be zero.

`duration` is an integer that indicates the duration of this track (in the timescale indicated in the `MovieHeaderBox`) This duration field may be indefinite (all 1s) when either there is no edit list and the `MediaHeaderBox` duration is indefinite (i.e. all 1s), or when an indefinitely repeated edit list is desired (see [subclause 8.6.6](#) for repeated edits).. If there is no edit list and the duration is not indefinite, then the duration shall be equal to the media duration given in the `MediaHeaderBox`.

converted into the timescale in the `MovieHeaderBox`. Otherwise the value of this field is equal to the sum of the durations of all of the track's edits (possibly including repetitions).

`layer` specifies the front-to-back ordering of video tracks; tracks with lower numbers are closer to the viewer. 0 is the normal value, and -1 would be in front of track 0, and so on.

`alternate_group` is an integer that specifies a group or collection of tracks. If this field is 0 there is no information on possible relations to other tracks. If this field is not 0, it should be the same for tracks that contain alternate data for one another and different for tracks belonging to different such groups. Only one track within an alternate group should be played or streamed at any one time, and shall be distinguishable from other tracks in the group via attributes such as bitrate, codec, language, packet size etc. A group may have only one member.

`volume` is a fixed 8.8 value specifying the track's relative audio volume. Full volume is 1.0 (0x0100) and is the normal value. Its value is irrelevant for a purely visual track. Tracks may be composed by combining them according to their volume, and then using the overall `MovieHeaderBox` volume setting; or more complex audio composition (e.g. MPEG-4 BIFS) may be used.

`matrix` provides a transformation matrix for the video; (u,v,w) are restricted here to (0,0,1), hex (0,0,0x40000000).

`width` and `height` fixed-point 16.16 values are track-dependent as follows:

For text and subtitle tracks, they may, depending on the coding format, describe the suggested size of the rendering area. For such tracks, the value 0x0 may also be used to indicate that the data may be rendered at any size, that no preferred size has been indicated and that the actual size may be determined by the external context or by reusing the width and height of another track. For those tracks, the flag `track_size_is_aspect_ratio` may also be used.

For non-visual tracks (e.g. audio), they should be set to zero.

For all other tracks, they specify the track's visual presentation size. These need not be the same as the pixel dimensions of the images, which is documented in the sample description(s); all images in the sequence are scaled to this size, before any overall transformation of the track represented by the matrix. The pixel dimensions of the images are the default values.

8.3.3 Track reference box

8.3.3.1 Definition

Box Type: 'tref'
 Container: `TrackBox`
 Mandatory: No
 Quantity: Zero or one

This box includes a set of `TrackReferenceTypeBoxes`, each of which indicates, by its type, that the enclosing track has one of more references of that type. Each reference type shall occur at most once. Within each `TrackReferenceTypeBox` there is an array of `track_IDS`; within a given array, a given value shall occur at most once. Other structures in the file formats index through these arrays; index values start at 1.

Exactly one `TrackReferenceBox` can be contained within the `TrackBox`.

If this box is not present, the track is not referencing any other track in any way. The reference array is sized to fill the reference type box.

8.3.3.2 Syntax

```
aligned(8) class TrackReferenceBox extends Box('tref') {
    TrackReferenceTypeBox [];
}
aligned(8) class TrackReferenceTypeBox (unsigned int(32) reference_type) extends
Box(reference_type) {
    unsigned int(32) track_IDs[];
}
```

8.3.3.3 Semantics

The `TrackReferenceBox` contains `TrackReferenceTypeBoxes`. There shall be at most one `TrackReferenceTypeBox` of a given type in a `TrackReferenceBox`.

`track_IDs` is an array of integers providing the track identifiers of the referenced tracks or `track_group_id` values of the referenced track groups. Each value `track_IDs[i]`, where `i` is a valid index to the `track_IDs[]` array, is an integer that provides a reference from the containing track to the track with `track_ID` equal to `track_IDs[i]` or to the track group with both `track_group_id` equal to `track_IDs[i]` and `(flags & 1)` of `TrackGroupTypeBox` equal to 1. When a `track_group_id` value is referenced, the track reference applies to each track of the referenced track group individually unless stated otherwise in the semantics of particular track reference types. The value 0 shall not be present. In the array there shall be no duplicated value; however, a `track_ID` may appear in the array and also be a member of one or more track groups for which the `track_group_IDs` appear in the array. This means that in forming the list of tracks, after replacing `track_group_IDs` by the `track_IDs` of the tracks in those groups, there might be duplicate `track_IDs`. A `track_group_ID` shall not be used when the semantics of the reference requires that the reference be to a single track.

The `reference_type` shall be set to one of the following values, or a value registered or from a derived specification or registration:

- 'hint' the referenced track(s) contain the original media for this hint track.
- 'cdsc' links a descriptive or metadata track to the content which it describes
- 'font' this track uses fonts carried/defined in the referenced track.
- 'hind' indicates that the referenced track(s) may contain media data required for decoding of the track containing the track reference, i.e., it should only be used if the referenced hint track is used. The referenced tracks shall be hint tracks. The 'hind' dependency can, for example, be used for indicating the dependencies between hint tracks documenting layered IP multicast over RTP.
- 'vdep' this track contains auxiliary depth video information for the referenced video track.
- 'vplx' this track contains auxiliary parallax video information for the referenced video track.
- 'subt' this track contains subtitle, timed text or overlay graphical information for the referenced track or any track in the alternate group to which the track belongs, if any.
- 'thmb' this track contains thumbnail images for the referenced track. A thumbnail track shall not be linked to another thumbnail track with the 'thmb' item reference.
- 'aux1' this track contains auxiliary media for the indicated track (e.g. depth map or alpha plane for video).
- 'cdtg' describes the referenced media tracks and track groups collectively; the 'cdtg' track reference shall only be present in timed metadata tracks.
- 'shsc' links a shadow sync track to a main track; see [subclause 8.6.3](#)

NOTE 1 A track with reference type 'aux1' could have a coding dependency; its use is clarified by specifications that use it.

NOTE 2 When multiple track references would describe an auxiliary video track, derived specifications might constrain or recommend which track references are used. For example, derived specifications might constrain or recommend whether to use 'vdep' or 'aux1' or both for auxiliary depth video track.

NOTE 3 Other structures index through the array of track references and hence position and order of them can be significant.

NOTE 4 A timed metadata track containing 'cdsc' track reference to a `track_group_id` value describes each track in the track group individually.

8.3.4 Track group box

8.3.4.1 Definition

Box Type: 'trgr'
 Container: `TrackBox`
 Mandatory: No
 Quantity: Zero or one

This box enables indication of groups of tracks, where each group shares a particular characteristic or the tracks within a group have a particular relationship. The box contains zero or more boxes, and the particular characteristic or the relationship is indicated by the box type of the contained boxes. The contained boxes include an identifier, which can be used to conclude the tracks belonging to the same track group. The tracks that contain the same type of a contained box within the `TrackGroupBox` and have the same identifier value within these contained boxes belong to the same track group.

Track groups shall not be used to indicate dependency relationships between tracks. Instead, the `TrackReferenceBox` is used for such purposes.

(`flags & 1`) equal to 1 in a `TrackGroupTypeBox` of a particular `track_group_type` indicates that `track_group_id` in that `TrackGroupTypeBox` is not equal to any `track_ID` value and is not equal to `track_group_id` of any other `TrackGroupTypeBox` with a different `track_group_type`. When (`flags & 1`) is equal to 1 in a `TrackGroupTypeBox` with particular values of `track_group_type` and `track_group_id`, (`flags & 1`) shall be equal to 1 in all `TrackGroupTypeBoxes` of the same values of `track_group_type` and `track_group_id`, respectively.

8.3.4.2 Syntax

```
aligned(8) class TrackGroupBox extends Box('trgr') {
}
aligned(8) class TrackGroupTypeBox(unsigned int(32) track_group_type) extends
FullBox(track_group_type, version = 0, flags = 0)
{
    unsigned int(32) track_group_id;
    // the remaining data may be specified
    // for a particular track_group_type
}
```

8.3.4.3 Semantics

`track_group_type` indicates the `grouping_type` and shall be set to one of the following values, or a value registered, or a value from a derived specification or registration:

'msrc' indicates that this track belongs to a multi-source presentation. Specified in [8.3.4.4.1](#).

'ster' indicates that this track is either the left or right view of a stereo pair suitable for playback on a stereoscopic display. Specified in [8.3.4.4.2](#).

The pair of `track_group_id` and `track_group_type` identifies a track group within the file. The tracks that contain a particular `TrackGroupBox` having the same value of `track_group_id` and `track_group_type` belong to the same track group.

8.3.4.4 Track group definitions

8.3.4.4.1 Multi-source presentation

`track_group_type` equal to 'msrc' indicates that this track belongs to a multi-source presentation. The tracks that have the same value of `track_group_id` within a `TrackGroupBox` of `track_group_type` 'msrc' are mapped as being originated from the same source. For example, a recording of a video telephony call may have both audio and video for both participants, and the value of `track_group_id` associated with the audio track and the video track of one participant differs from value of `track_group_id` associated with the tracks of the other participant.

8.3.4.4.2 Stereoscopic pair

8.3.4.4.2.1 Definition

`TrackGroupBox` with `track_group_type` equal to 'ster' indicates that this track is either the left or right view of a stereo pair suitable for playback on a stereoscopic display.

The tracks that have the same value of `track_group_id` within `StereoVideoGroupBox` form a stereo pair, and there shall be no more than two of such tracks for the same value of `track_group_id`.

NOTE Usually there are two tracks indicated to be a stereo pair with the `StereoVideoGroupBox` having the same value of `track_group_id`. However, only one track can be associated with a stereo pair in specific cases. For example, the file can be edited in a manner that one of the tracks forming a stereo pair gets removed. In another example, only one of the tracks of a stereo pair is selected for transmission, e.g. using DASH.

8.3.4.4.2.2 Syntax

```
aligned(8) class StereoVideoGroupBox extends TrackGroupBox('ster')
{
    unsigned int(1) left_view_flag;
    bit(31) reserved;
}
```

8.3.4.4.2.3 Semantics

`left_view_flag` equal to 0 indicates the right view of a stereo pair, and `left_view_flag` equal to 1 indicates the left view of a stereo pair. When there are two tracks with the same value of `track_group_id`, the value of `left_view_flag` shall differ.

8.3.5 Track type box

8.3.5.1 Definition

Box Type: 'ttyp'

Container: `TrackBox`

Mandatory: No

Quantity: Zero or one

The payload of `TrackTypeBox` has the same syntax as the payload of `FileTypeBox`. The content of an instance of `TrackTypeBox` shall be such that it would apply as the content of `FileTypeBox`, if all other tracks of the file were removed and only the track containing this box, and the tracks it references by means of track references, remained in the file.

NOTE `TrackTypeBox` can be used in specifying media profiles or track-specific brands.

8.3.5.2 Syntax

```
aligned(8) class TrackTypeBox extends GeneralTypeBox ('ttyp')
{ }
```

8.4 Track media structure

8.4.1 Media box

8.4.1.1 Definition

Box Type: 'mdia'
Container: `TrackBox`
Mandatory: Yes
Quantity: Exactly one

The media declaration container contains all the objects that declare information about the media data within a track.

8.4.1.2 Syntax

```
aligned(8) class MediaBox extends Box('mdia') {
}
```

8.4.2 Media header box

8.4.2.1 Definition

Box Type: 'mdhd'
Container: `MediaBox`
Mandatory: Yes
Quantity: Exactly one

The media header declares overall information that is media-independent, and relevant to characteristics of the media in a track.

8.4.2.2 Syntax

```
aligned(8) class MediaHeaderBox extends FullBox('mdhd', version, 0) {
    if (version==1) {
        unsigned int(64)    creation_time;
        unsigned int(64)    modification_time;
        unsigned int(32)    timescale;
        unsigned int(64)    duration;
    } else { // version==0
        unsigned int(32)    creation_time;
        unsigned int(32)    modification_time;
        unsigned int(32)    timescale;
        unsigned int(32)    duration;
    }
    bit(1)    pad = 0;
    unsigned int(5)[3]    language;    // ISO-639-2/T language code
    unsigned int(16)    pre_defined = 0;
}
```

8.4.2.3 Semantics

`version` is an integer that specifies the version of this box (0 or 1)

`creation_time` is an integer that declares the creation time of the media in this track (in seconds since midnight, Jan. 1, 1904, in UTC time).

`modification_time` is an integer that declares the most recent time the media in this track was modified (in seconds since midnight, Jan. 1, 1904, in UTC time).

`timescale` is an integer that specifies the number of time units that pass in one second for this media. For example, a time coordinate system that measures time in sixtieths of a second has a time scale of 60.

`duration` is an integer that declares the duration of this media (in the scale of the `timescale`) and should be the largest composition timestamp plus the duration of that sample. If the duration cannot be determined then duration is set to all 1s.

NOTE The duration of an audio track may be smaller than the duration of the audio samples output by the decoder. This depends on the decoding process. The decoding of the last ISOBMFF sample of a track may produce additional audio samples that are not meant to be rendered.

`language` declares the language code for this media, as a packed three-character code defined in ISO 639-2. Each character is packed as the difference between its ASCII value and 0x60. Since the code is confined to being three lower-case letters, these values are strictly positive.

8.4.3 Handler reference box

8.4.3.1 Definition

Box Type: 'hdlr'

Container: `MediaBox` or `MetaBox`

Mandatory: Yes

Quantity: Exactly one

This box within a `MediaBox` declares media type of the track, and thus the process by which the media-data in the track is presented. For example, a format for which the decoder delivers video would be stored in a video track, identified by being handled by a video handler. The documentation of the storage of a media format identifies the media type which that format uses.

This box when present within a `MetaBox`, declares the structure or format of the `MetaBox` contents.

There is a general handler for metadata streams of any type; the specific format is identified by the sample entry, as for video or audio, for example.

8.4.3.2 Syntax

```
aligned(8) class HandlerBox extends FullBox('hdlr', version = 0, 0) {
    unsigned int(32)    pre_defined = 0;
    unsigned int(32)    handler_type;
    const unsigned int(32)[3] reserved = 0;
    utf8string         name;
}
```

8.4.3.3 Semantics

`version` is an integer that specifies the version of this box

`handler_type`

- when present in a `MediaBox`, contains a value as defined in [Clause 12](#), or a value from a derived specification, or registration.
- when present in a `MetaBox`, contains an appropriate value to indicate the format of the `MetaBox` contents. The value 'null' can be used in the primary `MetaBox` to indicate that it is merely being used to hold resources.

`name` gives a human-readable name for the track type (for debugging and inspection purposes).

8.4.4 Media information box

8.4.4.1 Definition

Box Type: 'minf'
Container: MediaBox
Mandatory: Yes
Quantity: Exactly one

This box contains all the objects that declare characteristic information of the media in the track.

8.4.4.2 Syntax

```
aligned(8) class MediaInformationBox extends Box('minf') {  
}
```

8.4.5 Media information header boxes

8.4.5.1 Definition

There is a different media information header for each track type (corresponding to the media handler-type); the matching header shall be present, which may be one of those defined in [Clause 12](#), or one defined in a derived specification.

The type of media header is used is determined by the definition of the media type and shall match the media handler.

8.4.5.2 Null media header box

8.4.5.2.1 Definition

Box Types: 'nmhd'
Container: MediaInformationBox
Mandatory: Yes
Quantity: Exactly one specific media header shall be present

Streams for which no specific media header is identified use a NullMediaHeaderBox, as defined here.

8.4.5.2.2 Syntax

```
aligned(8) class NullMediaHeaderBox  
    extends FullBox('nmhd', version = 0, flags) {  
}
```

8.4.5.2.3 Semantics

version - is an integer that specifies the version of this box.

flags - is a 24-bit integer with flags (currently all zero).

8.4.6 Extended language tag

8.4.6.1 Definition

Box Type: 'elng'
Container: MediaBox
Mandatory: No
Quantity: Zero or one

The `ExtendedLanguageBox` represents media language information, and shall contain a code in conformance with IETF BCP 47. It is an optional peer of the `MediaHeaderBox`, and shall occur after the `MediaHeaderBox`.

The extended language tag can provide better language information than the language field in the `MediaHeaderBox`, including information such as region, script, variation, and so on, as parts (or subtags).

The `ExtendedLanguageBox` is optional, and if it is absent the media language should be used. The extended language tag overrides the media language if they are not consistent.

For best compatibility with earlier players, if an extended language tag is specified, the most compatible language code should be specified in the language field of the `MediaHeaderBox` (for example, "eng" if the extended language tag is "en-UK"). If there is no reasonably compatible tag, the packed form of 'und' can be used.

8.4.6.2 Syntax

```
aligned(8) class ExtendedLanguageBox extends FullBox('eLng', 0, 0) {
    utf8string    extended_language;
}
```

8.4.6.3 Semantics

`extended_language` contains a IETF BCP 47 compliant language tag string, such as "en-US", "fr-FR", or "zh-CN".

8.5 Sample tables

8.5.1 Sample table box

8.5.1.1 Definition

Box Type: 'stbl'

Container: `MediaInformationBox`

Mandatory: Yes

Quantity: Exactly one

The sample table contains all the time and data indexing of the media samples in a track. Using the tables here, it is possible to locate samples in time, determine their type (e.g. I-frame or not), and determine their size, container, and offset into that container.

If the track that contains the `SampleTableBox` references no data, then the `SampleTableBox` does not need to contain any sub-boxes (this is not a very useful media track).

If the track that the `SampleTableBox` is contained in does reference data, then the following sub-boxes are required: `SampleDescriptionBox`, `SampleSizeBox` (or `CompactSampleSizeBox`), `SampleToChunkBox`, and `ChunkOffsetBox` (or `ChunkLargeOffsetBox`). Further, the `SampleDescriptionBox` shall contain at least one entry. A `SampleDescriptionBox` is required because it contains the data reference index field which indicates which `DataEntry` to use to retrieve the media samples. Without the `SampleDescriptionBox`, it is not possible to determine where the media samples are stored. The `SyncSampleBox` is optional.

The `SyncSampleBox` should be present in the `SampleTableBox` if some samples in the track, including any track fragments, are non sync samples, but the flag `sample_is_non_sync_sample` of samples in track fragments is valid and describes the samples, even if the `SyncSampleBox` is not present. If the track is not fragmented and the `SyncSampleBox` is not present, all samples in the track are sync samples.

[A.9](#) provides a narrative description of random access using the structures defined in the `SampleTableBox`.

8.5.1.2 Syntax

```
aligned(8) class SampleTableBox extends Box('stbl') {
}
```

8.5.2 Sample description box

8.5.2.1 Definition

Box Types: 'std'

Container: SampleTableBox

Mandatory: Yes

Quantity: Exactly one

The sample description table gives detailed information about the coding type used, and any initialization information needed for that coding. The syntax of the sample entry used is determined by both the format field and the media handler type.

The information stored in the SampleDescriptionBox after the entry-count is both track-type specific as documented here, and can also have variants within a track type (e.g. different codings may use different specific information after some common fields, even within a video track).

Which type of sample entry form is used is determined by the media handler, using a suitable form, such as one defined in [Clause 12](#), or defined in a derived specification, or registration.

Multiple descriptions may be used within a track.

NOTE 1 Though the count is 32 bits, the number of items is usually much fewer, and is restricted by the fact that the reference index in the sample table is only 16 bits

If the 'format' field of a SampleEntry is unrecognized, neither the sample description itself, nor the associated media samples, shall be decoded.

NOTE 2 The format field of a SampleEntry is restricted in this document not to contain the character '.' (dot). Derived specifications are encouraged to avoid using it as well as any character that requires encoding of the "codecs" parameter; see [Annex K](#).

Derived specifications deriving Sample Entry classes listed in the table of [8.12.1](#) should be extremely careful. Derivation by adding boxes at the end of the class should be preferred as it preserves Sample Entry parsing and does not require a new 'encX' value. Adding a new field to a class will not allow for the use of the associated 'encX' scheme for parsing reasons. A new 'encX' scheme will have to be defined for signaling encrypted stream based on that derived class.

The definition of sample entries specifies boxes in a particular order, and this is usually also followed in derived specifications. For maximum compatibility, writers should construct files respecting the order both within specifications and as implied by the inheritance, whereas readers should be prepared to accept any box order.

All SampleEntry boxes may contain "extra boxes" not explicitly defined in the box syntax of this or derived specifications. When present, such boxes shall follow all defined fields and should follow any defined contained boxes. Decoders shall presume a sample entry box could contain extra boxes and shall continue parsing as though they are present until the containing box length is exhausted.

An optional BitRateBox may be present in any SampleEntry to signal the bit rate information of a stream. This can be used for buffer configuration.

All string fields shall be of type utf8string and null-terminated, even if unused. "Optional" means there is at least one null byte.

Entries that identify the format by MIME type, such as a TextSubtitleSampleEntry, TextMetaDataSampleEntry, or SimpleTextSampleEntry, all of which contain a MIME type, may be used to identify the format of streams for which a MIME type applies. A MIME type applies if the contents of the

string in the optional configuration box (without its null termination), followed by the contents of a set of samples, starting with a sync sample and ending at the sample immediately preceding a sync sample, are concatenated in their entirety, and the result meets the decoding requirements for documents of that MIME type. Non-sync samples should be used only if that format specifies the behaviour of 'progressive decoding', and then the sample times indicate when the results of such progressive decoding should be presented (according to the media type).

NOTE 3 The samples in a track that is all sync samples are therefore each a valid document for that MIME type.

In some classes derived from `SampleEntry`, `namespace` and `schema_location` are used both to identify the XML document content and to declare "brand" or profile compatibility. Multiple namespace identifiers indicate that the track conforms to the specification represented by each of the identifiers, some of which may identify supersets of the features present. A decoder should be able to decode all the namespaces in order to be able to decode and present correctly the media associated with this sample entry.

NOTE 4 Additionally, namespace identifiers might represent performance constraints, such as limits on document size, font size, drawing rate, etc., as well as syntax constraints, such as features that are not permitted or ignored.

8.5.2.2 Syntax

```
aligned(8) abstract class SampleEntry (unsigned int(32) format)
    extends Box(format){
    const unsigned int(8)[6] reserved = 0;
    unsigned int(16) data_reference_index;
}
class BitRateBox extends Box('btrt'){
    unsigned int(32) bufferSizeDB;
    unsigned int(32) maxBitrate;
    unsigned int(32) avgBitrate;
}
aligned(8) class SampleDescriptionBox ()
    extends FullBox('stsd', version, 0){
    int i ;
    unsigned int(32) entry_count;
    for (i = 1 ; i <= entry_count ; i++){
        SampleEntry(); // an instance of a class derived from SampleEntry
    }
}
```

8.5.2.3 Semantics

`version` is set to zero. A version number of 1 shall be treated as a version of 0.

`entry_count` is an integer that gives the number of entries in the following table

`SampleEntry` is the appropriate sample entry.

`data_reference_index` is an integer that contains the index of the `DataEntry` to use to retrieve data associated with samples that use this sample description. Data entries are stored in `DataReferenceBoxes`. The index ranges from 1 to the number of data entries.

`bufferSizeDB` gives the size of the decoding buffer for the elementary stream in bytes.

`maxBitrate` gives the maximum rate in bits/second over any window of one second; this is a measured value for stored content, or a value that a stream is configured not to exceed; the stream shall not exceed this bitrate.

`avgBitrate` gives the average rate in bits/second of the stream; this is a measured value (cumulative over the entire presentation) for stored content, or the configured target average bitrate for a stream.

8.5.3 Degradation priority box

8.5.3.1 Definition

Box Type: 'stdp'
 Container: SampleTableBox
 Mandatory: No.
 Quantity: Zero or one.

This box contains the degradation priority of each sample. The values are stored in the table, one for each sample. The size of the table, `sample_count` is taken from the `sample_count` in the `SampleSizeBox`. Specifications derived from this define the exact meaning and acceptable range of the priority field.

8.5.3.2 Syntax

```
aligned(8) class DegradationPriorityBox
  extends FullBox('stdp', version = 0, 0) {
    int i;
    for (i=0; i < sample_count; i++) {
        unsigned int(16) priority;
    }
}
```

8.5.3.3 Semantics

`version` - is an integer that specifies the version of this box.

`priority` - is integer specifying the degradation priority for each sample.

8.5.4 Sample scale box

This box has been deprecated and is no longer defined in this document.

8.6 Track time structures

8.6.1 Time to sample boxes

8.6.1.1 Definition

The composition timestamps (CT) and decoding timestamps (DT) of samples are derived from the time to sample boxes, of which there are two types. The decoding timestamp is defined by the `TimeToSampleBox`, which documents the sample duration, that is, the difference between the decoding timestamp of the following sample and the sample at hand. The composition timestamps are defined in the `CompositionOffsetBox` as time offsets from decoding timestamps. If the composition and decoding timestamps are identical for every sample in the track, then only the `TimeToSampleBox` is required; the `CompositionOffsetBox` shall not be present and all composition offsets are defined to be zero.

The `TimeToSampleBox` shall give non-zero durations for all samples with the possible exception of the last one. Durations in the `TimeToSampleBox` are strictly positive (non-zero), except for the very last entry, which may be zero. This rule derives from the rule that there shall not be two samples in a stream with the same decoding timestamp. Great care must be taken when adding samples to a stream, that the sample that was previously last may need to have a non-zero duration established, in order to observe this rule. One approach in the case where the duration of the last sample is indeterminate is to use an arbitrary small value and a 'dwell' edit.

Some coding systems may allow samples that are used only for reference and not output (e.g. a non-displayed reference frame in video). When any such non-output sample is present in a track, the following applies:

- 1) A non-output sample shall be given a composition timestamp which is outside the time-range of the samples that are output.
- 2) An edit list shall be used to exclude the composition times of the non-output samples.
- 3) When the track includes a `CompositionOffsetBox`,
 - a) version 1 of the `CompositionOffsetBox` shall be used,
 - b) the value of `sample_offset` shall be set equal to the most negative number possible (for 32-bit values, -2^{31}) for each non-output sample,
 - c) the `CompositionToDecodeBox` should be contained in the `SampleTableBox` of the track, and
 - d) when the `CompositionToDecodeBox` is present for the track, the value of `leastDecodeToDisplayDelta` field in the box shall be equal to the smallest composition offset in the `CompositionOffsetBox` excluding the `sample_offset` values for non-output samples.

NOTE Thus, `leastDecodeToDisplayDelta` is greater than -2^{31} .

In the example in [Table 2](#) and [Table 3](#), there is a sequence of I, P, and B frames, each with a sample duration of 10. The samples are stored as follows, with the indicated values for their sample durations and composition time offsets (the actual composition timestamp, CT, and decoding timestamp, DT, are given for reference). The re-ordering occurs because the predicted P frames must be decoded before the bi-directionally predicted B frames. The value of DT for a sample is always the sum of the durations of the preceding samples. Thus, in the absence of composition offsets, the total of the sample durations is the duration of the media in this track.

Table 2 — Closed GOP example

GOP	/--	---	---	---	---	---	--\	/--	---	---	---	---	---	--\
	I1	P4	B2	B3	P7	B5	B6	I8	P11	B9	B10	P14	B12	B13
DT	0	10	20	30	40	50	60	70	80	90	100	110	120	130
CT	10	40	20	30	70	50	60	80	110	90	100	140	120	130
Duration	10	10	10	10	10	10	10	10	10	10	10	10	10	10
Composition offset	10	30	0	0	30	0	0	10	30	0	0	30	0	0

Table 3 — Open GOP example

GOP	/--	--	--	--	--	--\	/-	--	--	--	---	--\
	I3	B1	B2	P6	B4	B5	I9	B7	B8	P12	B10	B11
DT	0	10	20	30	40	50	60	70	80	90	100	110
CT	30	10	20	60	40	50	90	70	80	120	100	110
Duration	10	10	10	10	10	10	10	10	10	10	10	10
Composition offset	30	0	0	30	0	0	30	0	0	30	0	0

8.6.1.2 Decoding time to sample box

8.6.1.2.1 Definition

Box Type: 'stts'

Container: `SampleTableBox`

Mandatory: Yes
Quantity: Exactly one

This box contains a compact version of a table that allows indexing from decoding timestamp to sample number. Other tables give sample sizes and pointers, from the sample number. Each entry in the table gives the number of consecutive samples with the same sample duration, and that sample duration. By adding the sample durations a complete time-to-sample map may be built.

The `TimeToSampleBox` contains sample durations, the differences in decoding timestamps (DT):

$$DT[n+1] = DT[n] + sample_delta[n]$$

The sample entries are ordered by decoding timestamps; therefore all the values of `sample_delta` shall be non-negative.

The DT axis has a zero origin; $DT[i] = \text{SUM}[\text{for } j=0 \text{ to } i-1 \text{ of } sample_delta[j]]$, and in the absence of composition offsets, the sum of all sample durations gives the duration of the media in the track (not mapped to the overall timescale, and not considering any edit list).

8.6.1.2.2 Syntax

```
aligned(8) class TimeToSampleBox
  extends FullBox('stts', version = 0, 0) {
    unsigned int(32)  entry_count;
    int i;
    for (i=0; i < entry_count; i++) {
      unsigned int(32)  sample_count;
      unsigned int(32)  sample_delta;
    }
  }
```

For example with [Table 2](#), the entry would be:

Sample count	Sample-delta
14	10

8.6.1.2.3 Semantics

- `version` - is an integer that specifies the version of this box.
- `entry_count` - is an integer that gives the number of entries in the following table.
- `sample_count` - is an integer that counts the number of consecutive samples that have the given duration.
- `sample_delta` - is an integer that gives the difference between the decoding timestamp of the next sample and this one, in the time-scale of the media.

8.6.1.3 Composition time to sample box

8.6.1.3.1 Definition

Box Type: 'ctts'
Container: `SampleTableBox`
Mandatory: No
Quantity: Zero or one

This box provides the offset between decoding timestamp and composition timestamp. In version 0 of this box the decoding timestamp must be less than the composition timestamp, and the offsets are expressed as unsigned numbers such that if CT is the composition timestamp, $CT[n] = DT[n] + sample_offset[n]$.