
**Information technology — JPEG 2000
image coding system —**

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

*Technologies de l'information — Système de codage d'images JPEG
2000 —*

Partie 12: Format ISO de base pour les fichiers médias



Reference number
ISO/IEC 15444-12:2012(E)

Documents
Document Sharing Hub
© ISO/IEC 2012



COPYRIGHT PROTECTED DOCUMENT

© ISO/IEC 2012

All rights reserved. Unless otherwise specified, no part of this publication may be reproduced or utilized in any form or by any means, electronic or mechanical, including photocopying and microfilm, without permission in writing from either ISO at the address below or ISO's member body in the country of the requester.

ISO copyright office
Case postale 56 • CH-1211 Geneva 20
Tel. + 41 22 749 01 11
Fax + 41 22 749 09 47
E-mail copyright@iso.org
Web www.iso.org

Published in Switzerland

Contents

Page

Foreword	ix
Introduction.....	xi
1 Scope.....	1
2 Normative references	1
3 Definitions.....	2
3.1 Terms and definitions	2
3.2 Abbreviated terms.....	4
4 Object-structured File Organization.....	4
4.1 File Structure	4
4.2 Object Structure	4
4.3 File Type Box	5
5 Design Considerations	6
5.1 Usage.....	6
5.1.1 Introduction	6
5.1.2 Interchange	6
5.1.3 Content Creation	7
5.1.4 Preparation for streaming	8
5.1.5 Local presentation	8
5.1.6 Streamed presentation	8
5.2 Design principles	8
6 ISO Base Media File organization	9
6.1 Presentation structure	9
6.1.1 File Structure	9
6.1.2 Object Structure	9
6.1.3 Meta Data and Media Data.....	9
6.1.4 Track Identifiers	10
6.2 Metadata Structure (Objects)	10
6.2.1 Box.....	10
6.2.2 Data Types and fields	10
6.2.3 Box Order.....	11
6.2.4 URIs as type indicators	13
6.3 Brand Identification	14
7 Streaming Support.....	14
7.1 Handling of Streaming Protocols	14
7.2 Protocol 'hint' tracks	14
7.3 Hint Track Format	15
8 Box Structures.....	16
8.1 File Structure and general boxes	16
8.1.1 Media Data Box	16
8.1.2 Free Space Box	17
8.1.3 Progressive Download Information Box.....	17
8.2 Movie Structure	18
8.2.1 Movie Box	18
8.2.2 Movie Header Box	18
8.3 Track Structure.....	19
8.3.1 Track Box	19
8.3.2 Track Header Box.....	20
8.3.3 Track Reference Box	21

8.3.4	Track Group Box	22
8.4	Track Media Structure	23
8.4.1	Media Box	23
8.4.2	Media Header Box	23
8.4.3	Handler Reference Box	24
8.4.4	Media Information Box	25
8.4.5	Media Information Header Boxes	25
8.5	Sample Tables	27
8.5.1	Sample Table Box	27
8.5.2	Sample Description Box	28
8.5.3	Degradation Priority Box	33
8.5.4	Sample Scale Box	34
8.6	Track Time Structures	34
8.6.1	Time to Sample Boxes	34
8.6.2	Sync Sample Box	38
8.6.3	Shadow Sync Sample Box	39
8.6.4	Independent and Disposable Samples Box	40
8.6.5	Edit Box	41
8.6.6	Edit List Box	41
8.7	Track Data Layout Structures	43
8.7.1	Data Information Box	43
8.7.2	Data Reference Box	43
8.7.3	Sample Size Boxes	44
8.7.4	Sample To Chunk Box	45
8.7.5	Chunk Offset Box	46
8.7.6	Padding Bits Box	47
8.7.7	Sub-Sample Information Box	48
8.7.8	Sample Auxiliary Information Sizes Box	49
8.7.9	Sample Auxiliary Information Offsets Box	50
8.8	Movie Fragments	51
8.8.1	Movie Extends Box	51
8.8.2	Movie Extends Header Box	52
8.8.3	Track Extends Box	52
8.8.4	Movie Fragment Box	53
8.8.5	Movie Fragment Header Box	53
8.8.6	Track Fragment Box	54
8.8.7	Track Fragment Header Box	54
8.8.8	Track Fragment Run Box	55
8.8.9	Movie Fragment Random Access Box	56
8.8.10	Track Fragment Random Access Box	57
8.8.11	Movie Fragment Random Access Offset Box	58
8.8.12	Track fragment decode time	58
8.8.13	Level Assignment Box	59
8.8.14	Sample Auxiliary Information in Movie Fragments	61
8.9	Sample Group Structures	61
8.9.1	Introduction	61
8.9.2	Sample to Group Box	62
8.9.3	Sample Group Description Box	63
8.9.4	Representation of group structures in Movie Fragments	64
8.10	User Data	65
8.10.1	User Data Box	65
8.10.2	Copyright Box	65
8.10.3	Track Selection Box	66
8.11	Metadata Support	68
8.11.1	The Meta box	68
8.11.2	XML Boxes	69
8.11.3	The Item Location Box	69
8.11.4	Primary Item Box	71
8.11.5	Item Protection Box	72
8.11.6	Item Information Box	72

8.11.7	Additional Metadata Container Box	74
8.11.8	Metabox Relation Box	75
8.11.9	URL Forms for meta boxes	75
8.11.10	Static Metadata	76
8.11.11	Item Data Box	77
8.11.12	Item Reference Box	77
8.11.13	Auxiliary video metadata	78
8.12	Support for Protected Streams	78
8.12.1	Protection Scheme Information Box	79
8.12.2	Original Format Box	80
8.12.3	IPMPInfoBox	80
8.12.4	IPMP Control Box	80
8.12.5	Scheme Type Box	80
8.12.6	Scheme Information Box	81
8.13	File Delivery Format Support	81
8.13.1	Introduction	81
8.13.2	FD Item Information Box	82
8.13.3	File Partition Box	82
8.13.4	FEC Reservoir Box	84
8.13.5	FD Session Group Box	84
8.13.6	Group ID to Name Box	85
8.13.7	File Reservoir Box	86
8.14	Sub tracks	86
8.14.1	Introduction	86
8.14.2	Backward compatibility	87
8.14.3	Sub Track box	87
8.14.4	Sub Track Information box	87
8.14.5	Sub Track Definition box	88
8.14.6	Sub Track Sample Group box	89
8.15	Post-decoder requirements on media	89
8.15.1	General	89
8.15.2	Transformation	89
8.15.3	Restricted Scheme Information box	90
8.15.4	Scheme for stereoscopic video arrangements	90
8.16	Segments	92
8.16.1	Introduction	92
8.16.2	Segment Type Box	92
8.16.3	Segment Index Box	93
8.16.4	Subsegment Index Box	96
8.16.5	Producer Reference Time Box	98
9	Hint Track Formats	99
9.1	RTP and SRTP Hint Track Format	99
9.1.1	Introduction	99
9.1.2	Sample Description Format	99
9.1.3	Sample Format	101
9.1.4	SDP Information	103
9.1.5	Statistical Information	104
9.2	ALC/LCT and FLUTE Hint Track Format	105
9.2.1	Introduction	105
9.2.2	Design principles	105
9.2.3	Sample Description Format	107
9.2.4	Sample Format	107
9.3	MPEG-2 Transport Hint Track Format	110
9.3.1	Introduction	110
9.3.2	Design Principles	111
9.3.3	Sample Description Format	112
9.3.4	Sample Format	114
9.3.5	Protected MPEG 2 Transport Stream Hint Track	116
9.4	RTP, RTCP, SRTP and SRTCP Reception Hint Tracks	117

9.4.1	RTP Reception Hint Track	117
9.4.2	RTCP Reception Hint Track.....	120
9.4.3	SRTCP Reception Hint Track.....	121
9.4.4	SRTCP Reception Hint Tracks	123
9.4.5	Protected RTP Reception Hint Track.....	124
9.4.6	Recording Procedure	124
9.4.7	Parsing Procedure.....	124
10	Sample Groups	124
10.1	Random Access Recovery Points	124
10.2	Rate Share Groups	125
10.2.1	Introduction	125
10.2.2	Rate Share Sample Group Entry	126
10.2.3	Relationship between tracks	127
10.2.4	Bitrate allocation.....	128
10.3	Alternative Startup Sequences	128
10.3.1	Definition	128
10.3.2	Syntax	129
10.3.3	Semantics	129
10.3.4	Examples	129
10.4	Random Access Point (RAP) Sample Grouping	131
10.4.1	Definition	131
10.4.2	Syntax	131
10.4.3	Semantics	131
10.5	Temporal level sample grouping	131
10.5.1	Definition	131
10.5.2	Syntax	132
10.5.3	Semantics	132
11	Extensibility	132
11.1	Objects	132
11.2	Storage formats	133
11.3	Derived File formats	133
Annex A	(informative) Overview and Introduction.....	134
A.1	Section Overview	134
A.2	Core Concepts	134
A.3	Physical structure of the media	134
A.4	Temporal structure of the media.....	135
A.5	Interleave	135
A.6	Composition	135
A.7	Random access.....	136
A.8	Fragmented movie files.....	136
Annex B	(informative) Patent Statements.....	138
Annex C	(informative) Guidelines on deriving from this specification.....	139
C.1	Introduction	139
C.2	General Principles	139
C.2.1	General.....	139
C.2.2	Base layer operations.....	139
C.3	Boxes	140
C.4	Brand Identifiers	140
C.4.1	Introduction	140
C.4.2	Usage of the Brand	141
C.4.3	Introduction of a new brand	141
C.4.4	Player Guideline.....	141
C.4.5	Authoring Guideline	142
C.4.6	Example	142
C.5	Storage of new media types	142
C.6	Use of Template fields.....	143
C.7	Tracks	143

C.7.1	Data Location	143
C.7.2	Time	143
C.7.3	Media Types	144
C.7.4	Coding Types	144
C.7.5	Sub-sample information	144
C.7.6	Sample Dependency	144
C.7.7	Sample Groups	144
C.7.8	Track-level	144
C.7.9	Protection	145
C.8	Construction of fragmented movies	145
C.9	Meta-data	146
C.10	Registration	146
C.11	Guidelines on the use of sample groups, timed metadata tracks, and sample auxiliary information	146
Annex D	(informative) Registration Authority	148
D.1	Code points to be registered	148
D.2	Procedure for the request of an MPEG-4 registered identifier value	148
D.3	Responsibilities of the Registration Authority	149
D.4	Contact information for the Registration Authority	149
D.5	Responsibilities of Parties Requesting a RID	149
D.6	Appeal Procedure for Denied Applications	150
D.7	Registration Application Form	150
D.7.1	Contact Information of organization requesting a RID	150
D.7.2	Request for a specific RID	150
D.7.3	Short description of RID that is in use and date system was implemented	151
D.7.4	Statement of an intention to apply the assigned RID	151
D.7.5	Date of intended implementation of the RID	151
D.7.6	Authorized representative	151
D.7.7	For official use of the Registration Authority	151
Annex E	(normative) File format brands	152
E.1	Introduction	152
E.2	The 'isom' brand	153
E.3	The 'avc1' brand	154
E.4	The 'iso2' brand	154
E.5	The 'mp71' brand	155
E.6	The 'iso3' brand	155
E.7	The 'iso4' brand	155
E.8	The 'iso5' brand	156
E.9	The 'iso6' brand	156
Annex F	(informative) Document Cross-Reference	157
Annex G	(informative) URI-labelled metadata forms	159
G.1	UUID-labelled metadata	159
G.2	ISO OID-labelled metadata	159
G.3	SMPTE-labelled metadata	159
Annex H	(informative) Processing of RTP streams and reception hint tracks	161
H.1	Introduction	161
H.1.1	Overview	161
H.1.2	Structure	161
H.1.3	Terms and definitions	161
H.2	Synchronization of RTP streams	161
H.3	Recording of RTP streams	162
H.3.1	Introduction	162
H.3.2	Compensation for unequal starting for position of received RTP streams	164
H.3.3	Recording of SDP	165
H.3.4	Creation of a sample within an RTP reception hint track	165
H.3.5	Representation of RTP timestamps	166
H.3.6	Recording operations to facilitate inter-stream synchronization in playback	169

H.3.7	Representation of reception times.....	170
H.3.8	Creation of media samples	171
H.3.9	Creation of hint samples referring to media samples.....	171
H.4	Playing of recorded RTP streams	171
H.4.1	Introduction	171
H.4.2	Preparation for the playback	172
H.4.3	Decoding of a sample within an RTP reception hint track	172
H.4.4	Lip synchronization	172
H.4.5	Random access.....	174
H.5	Re-sending recorded RTP streams.....	174
H.5.1	Introduction	174
H.5.2	Re-sending RTP packets.....	175
H.5.3	RTCP Processing.....	176
Annex I (normative)	Stream Access Points.....	177
I.1	Introduction.....	177
I.2	SAP properties.....	177
I.3	SAP types	177
Annex J (normative)	MIME Type Registration of Segments	179
J.1	Introduction.....	179
J.2	Registration.....	179
Bibliography		180

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. In the field of information technology, ISO and IEC have established a joint technical committee, ISO/IEC JTC 1.

International Standards are drafted in accordance with the rules given in the ISO/IEC Directives, Part 2.

The main task of the joint technical committee is to prepare International Standards. Draft International Standards adopted by the joint technical committee are circulated to national bodies for voting. Publication as an International Standard requires approval by at least 75 % of the national bodies casting a vote.

ISO/IEC 15444-12 was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee SC 29, *Coding of audio, picture, multimedia and hypermedia information*.

This fourth edition cancels and replaces the third edition (ISO/IEC 15444-12:2008) of which it constitutes a minor revision. It also incorporates the Amendment ISO/IEC 15444-12:2008/Amd.1:2009 and the Technical Corrigenda ISO/IEC 15444-12:2008/Cor.1:2008, ISO/IEC 15444-12:2008/Cor.2:2009, ISO/IEC 15444-12:2008/Cor.3:2009, and ISO/IEC 15444-12:2008/Cor.4:2011.

ISO/IEC 15444 consists of the following parts, under the general title *Information technology — JPEG 2000 image coding system*:

- *Part 1: Core coding system*
- *Part 2: Extensions*
- *Part 3: Motion JPEG 2000*
- *Part 4: Conformance testing*
- *Part 5: Reference software*
- *Part 6: Compound image file format*
- *Part 8: Secure JPEG 2000*
- *Part 9: Interactivity tools, APIs and protocols*
- *Part 10: Extensions for three-dimensional data*
- *Part 11: Wireless*
- *Part 12: ISO base media file format*
- *Part 13: An entry level JPEG 2000 encoder*
- *Part 14: XML structural representation and reference¹*

¹ To be published.

ISO/IEC 15444-12:2012(E)

This corrected version of ISO/IEC 15444-12:2012 incorporates the corrections made by ISO/IEC 15444-12:2008 draft Technical Corrigendum 5 (unpublished).

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.

It is intended that the ISO Base Media File Format shall be jointly maintained by WG1 and WG11. Consequently, a subdivision of work created ISO/IEC 15444-12 and ISO/IEC 14496-12 in order to document the ISO Base Media File Format and to facilitate the joint maintenance.

This technically identical text is published as ISO/IEC 14496-12 for MPEG-4, and as ISO/IEC 15444-12 for JPEG 2000, and reference to this specification should be made accordingly. The recommendation is to reference one, for example ISO/IEC 14496-12, and append to the reference a parenthetical comment identifying the other, for example "(technically identical to ISO/IEC 15444-12)".

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.

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

The holder of this patent right has assured the ISO and IEC that he is willing to negotiate licences under reasonable and non-discriminatory terms and conditions with applicants throughout the world. In this respect, the statement of the holder of this patent right is registered with the ISO and IEC. Information may be obtained from the companies listed in Annex B.

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

ISO (www.iso.org/patents) and IEC (<http://patents.iec.ch>) maintain on-line databases of patents relevant to their standards. Users are encouraged to consult the databases for the most up to date information concerning patents.

Information technology — JPEG 2000 image coding system —

Part 12: ISO base media file format

1 Scope

This part of ISO/IEC 15444 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.

This part of ISO/IEC 15444 is applicable to JPEG 2000, but its technical content is identical to that of ISO/IEC 14496-12, which is applicable to MPEG-4.

2 Normative references

The following documents, in whole or in part, are normatively referenced in this document and are indispensable for its application. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

ISO 639-2:1998, *Codes for the representation of names of languages — Part 2: Alpha-3 code*

ISO/IEC 9834-8:2005, *Information technology — Open Systems Interconnection — Procedures for the operation of OSI Registration Authorities: Generation and registration of Universally Unique Identifiers (UUIDs) and their use as ASN.1 Object Identifier components*

ISO/IEC 11578:1996, *Information technology — Open Systems Interconnection — Remote Procedure Call (RPC)*

ISO/IEC 14496-1:2010, *Information technology — Coding of audio-visual objects — Part 1: Systems*

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

ISO/IEC 14496-14, *Information technology — Coding of audio-visual objects — Part 14: MP4 file format*

ISO/IEC 15444-1, *Information technology — JPEG 2000 image coding system: Core coding system*

ISO/IEC 15444-3, *Information technology — JPEG 2000 image coding system: Motion JPEG 2000*

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 29199-2:2012, *Information technology — JPEG XR image coding system — Part 2: Image coding specification*

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

IETF RFC 2045, *Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies*, FREED, N. and BORENSTEIN, N., November 1996

IETF RFC 2046, *Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types*, FREED, N. and BORENSTEIN, N., November 1996

IETF RFC 3550, *RTP: A Transport Protocol for Real-Time Applications*, SCHULZRINNE, H. et al., July 2003.

IETF RFC 3711, *"The Secure Real-time Transport Protocol (SRTP)"*, BAUGHER, M. et al., March 2004

IETF RFC 5052, *Forward Error Correction (FEC) Building Block*, WATSON, M. et al., August 2007

IETF RFC 5905, *Network Time Protocol Version 4: Protocol and Algorithms Specification*, MILLS, D., et al, June 2010

SMIL 1.0 "Synchronized Multimedia Integration Language (SMIL) 1.0 Specification", <<http://www.w3.org/TR/REC-smil/>>

Rec. ITU-R TF.460-6, *Standard-frequency and time-signal emissions (Annex I for the definition of UTC.)*

3 Terms, definitions, and abbreviated terms

3.1 Terms and definitions

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

3.1.1

box

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

NOTE 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

container box

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

NOTE Container boxes are normally not derived from 'fullbox'

3.1.4

hint track

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

3.1.5

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

3.1.6**ISO Base Media File**

name of the files conforming to the file format described in this specification

3.1.7**leaf subsegment**

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

3.1.8**media data box**

box which can hold the actual media data for a presentation (‘mdat’)

3.1.9**movie box**

container box whose sub-boxes define the metadata for a presentation (‘moov’)

3.1.10**presentation**

one or more motion sequences, possibly combined with audio

3.1.11**random access point (RAP)**

sample in a track that starts at the ISAU of a SAP of type 1 or 2 or 3 as defined in Annex I

NOTE Informally, a sample, from which when decoding starts, the sample itself and all samples following in composition order can be correctly decoded.

3.1.12**random access recovery point**

sample in a track with presentation time equal to the TSAP of a SAP of type 4 as defined in Annex I

NOTE 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.

3.1.13**sample**

all the data associated with a single timestamp

NOTE 1 No two samples within a track can share the same time-stamp.

NOTE 2 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.

3.1.14**sample description**

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

3.1.15**sample table**

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

3.1.16**sync sample**

sample in a track that starts at the ISAU of a SAP of type 1 or 2 as defined in Annex I

NOTE 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.

3.1.17

segment

portion of an ISO base media file format file, consisting of either (a) a movie box, with its associated media data (if any) and other associated boxes or (b) one or more movie fragment boxes, with their associated media data, and other associated boxes

3.1.18

subsegment

time interval of a segment formed from movie fragment boxes, that is also a valid segment

3.1.19

track

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

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

3.2 Abbreviated terms

For the purposes of this document, the following abbreviated terms apply.

ALC	Asynchronous Layered Coding
FD	File Delivery
FDT	File Delivery Table
FEC	Forward Error Correction
FLUTE	File Delivery over Unidirectional Transport
IANA	Internet Assigned Numbers Authority
LCT	Layered Coding Transport
MBMS	Multimedia Broadcast/Multicast Service

4 Object-structured File Organization

4.1 File Structure

Files are formed as a series of objects, called boxes in this specification. 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 this section of this specification (all Object-Structured files) shall contain a File Type Box.

4.2 Object Structure

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 Media Data Box(es) need the 64-bit size.

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.

The definitions of boxes are given in the syntax description language (SDL) defined in MPEG-4 (see reference in Clause 2). Comments in the code fragments in this specification indicate informative material.

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.

```
aligned(8) class Box (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;
    }
}
```

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 is the last one in the file, and its contents extend to the end of the file (normally only used for a Media Data Box)
`type` identifies the box type; standard boxes use a compact type, which is normally four printable characters, to permit ease of identification, and is shown so in the boxes below. User extensions use an extended type; in this case, the type field is set to 'uuid'.

Boxes with an unrecognized type shall be ignored and skipped.

Many objects also contain a version number and flags field:

```
aligned(8) class FullBox(unsigned int(32) boxtype, unsigned int(8) v, bit(24) f)
    extends Box(boxtype) {
    unsigned int(8)    version = v;
    bit(24)           flags = f;
}
```

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

Boxes with an unrecognized version shall be ignored and skipped.

4.3 File Type Box

4.3.1 Definition

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

Files written to this version of this specification must contain a file-type box. For compatibility with an earlier version of this specification, files may be conformant to this specification and not contain a file-type box. Files with no file-type box should be read as if they contained an FTYP box with `Major_brand='mp41'`, `minor_version=0`, and the single compatible brand 'mp41'.

A media-file structured to this part of this specification 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 must 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 Movie Box, Media Data Box, or Free Space). 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 must not be 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 section of this specification does not define any brands. However, see subclause 6.3 below for brands for files conformant to the whole specification and not just this section. All file format brands defined in this specification are included in Annex E with a summary of which features they require.

4.3.2 Syntax

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

4.3.3 Semantics

This box identifies the specifications to which this file complies.

Each brand is a printable 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

5 Design Considerations

5.1 Usage

5.1.1 Introduction

The file format is intended to serve as a basis for a number of operations. In these various roles, it may be used in different ways, and different aspects of the overall design exercised.

5.1.2 Interchange

When used as an interchange format, the files would normally be self-contained (not referencing media in other files), contain only the media data actually used in the presentation, and not contain any information related to streaming. This will result in a small, protocol-independent, self-contained file, which contains the core media data and the information needed to operate on it.

The following diagram gives an example of a simple interchange file, containing two streams.

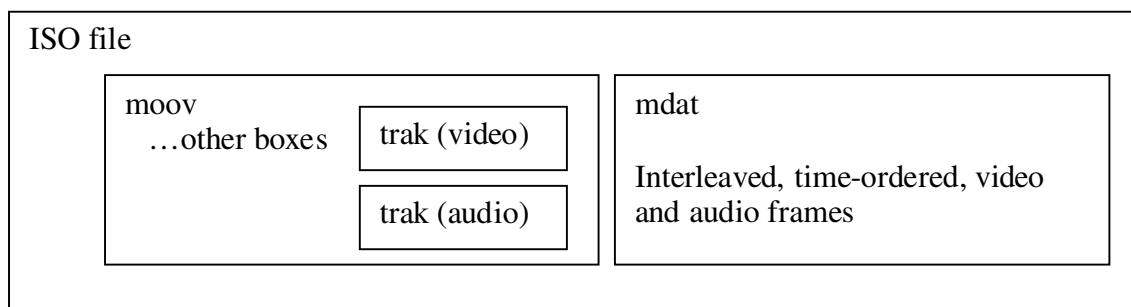


Figure 1 — Simple interchange file

5.1.3 Content Creation

During content creation, a number of areas of the format can be exercised to useful effect, particularly:

- the ability to store each elementary stream separately (not interleaved), possibly in separate files.
- the ability to work in a single presentation that contains media data and other streams (e.g. editing the audio track in the uncompressed format, to align with an already-prepared video track).

These characteristics mean that presentations may be prepared, edits applied, and content developed and integrated without either iteratively re-writing the presentation on disc – which would be necessary if interleave was required and unused data had to be deleted; and also without iteratively decoding and re-encoding the data – which would be necessary if the data must be stored in an encoded state.

In the following diagram, a set of files being used in the process of content creation is shown.

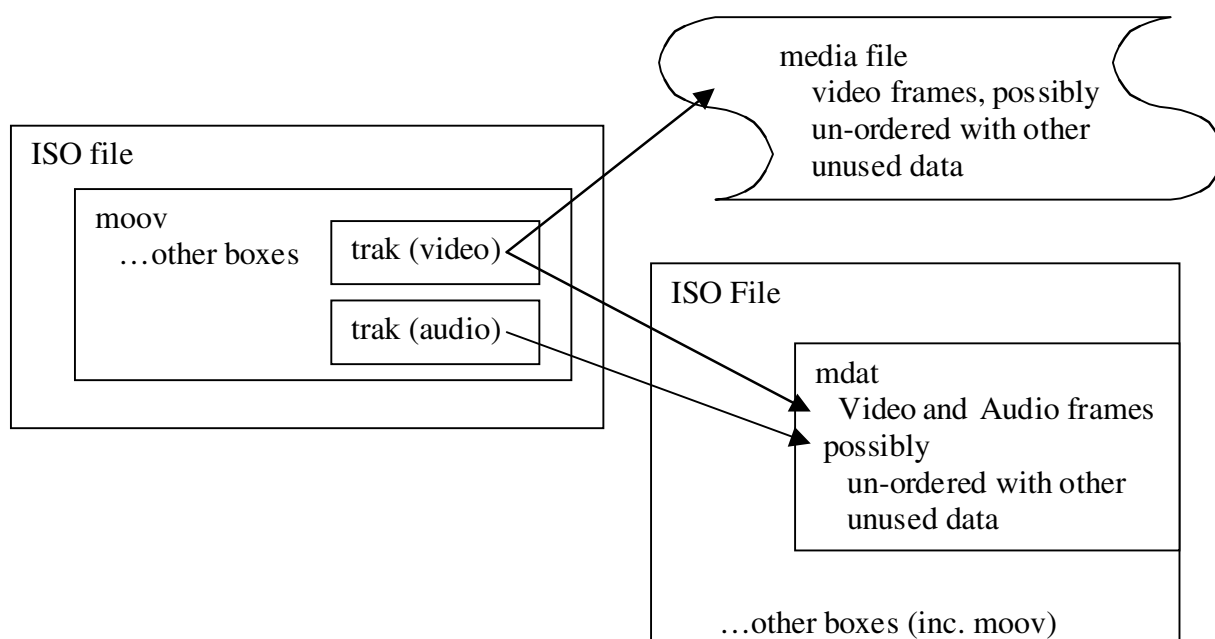


Figure 2 — Content Creation File

5.1.4 Preparation for streaming

When prepared for streaming, the file must contain information to direct the streaming server in the process of sending the information. In addition, it is helpful if these instructions and the media data are interleaved so that excessive seeking can be avoided when serving the presentation. It is also important that the original media data be retained unscathed, so that the files may be verified, or re-edited or otherwise re-used. Finally, it is helpful if a single file can be prepared for more than one protocol, so differing servers may use it over disparate protocols.

5.1.5 Local presentation

'Locally' viewing a presentation (i.e. directly from the file, not over a streamed interconnect) is an important application; it is used when a presentation is distributed (e.g. on CD or DVD ROM), during the process of development, and when verifying the content on streaming servers. Such local viewing must be supported, with full random access. If the presentation is on CD or DVD ROM, interleave is important as seeking may be slow.

5.1.6 Streamed presentation

When a server operates from the file to make a stream, the resulting stream must be conformant with the specifications for the protocol(s) used, and should contain no trace of the file-format information in the file itself. The server needs to be able to random access the presentation. It can be useful to re-use server content (e.g. to make excerpts) by referencing the same media data from multiple presentations; it can also assist streaming if the media data can be on read-only media (e.g. CD) and not copied, merely augmented, when prepared for streaming.

The following diagram shows a presentation prepared for streaming over a multiplexing protocol, only one hint track is required.

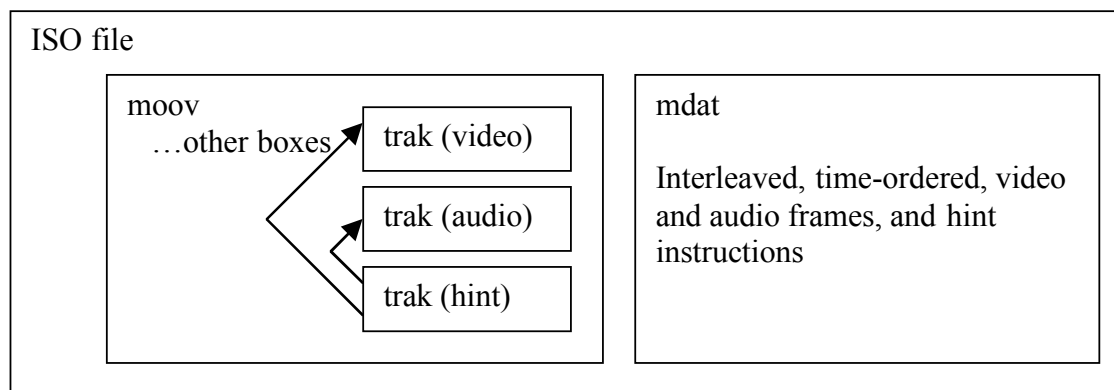


Figure 3 — Hinted Presentation for Streaming

5.2 Design principles

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.

Media-data is not 'framed' by the file format; the file format declarations that give the size, type and position of media data units are not physically contiguous with the media data. This makes it possible to subset the media-data, and to use it in its natural state, without requiring it to be copied to make space for framing. The metadata is used to describe the media data by reference, not by inclusion.

Similarly the protocol information for a particular streaming protocol does not frame the media data; the protocol headers are not physically contiguous with the media data. Instead, the media data can be included

by reference. This makes it possible to represent media data in its natural state, not favouring any protocol. It also makes it possible for the same set of media data to serve for local presentation, and for multiple protocols.

The protocol information is built in such a way that the streaming servers need to know only about the protocol and the way it should be sent; the protocol information abstracts knowledge of the media so that the servers are, to a large extent, media-type agnostic. Similarly the media-data, stored as it is in a protocol-unaware fashion, enables the media tools to be protocol-agnostic.

The file format does not require that a single presentation be in a single file. This enables both sub-setting and re-use of content. When combined with the non-framing approach, it also makes it possible to include media data in files not formatted to this specification (e.g. 'raw' files containing only media data and no declarative information, or file formats already in use in the media or computer industries).

The file format is based on a common set of designs and a rich set of possible structures and usages. The same format serves all usages; translation is not required. However, when used in a particular way (e.g. for local presentation), the file may need structuring in certain ways for optimal behaviour (e.g. time-ordering of the data). No normative structuring rules are defined by this specification, unless a restricted profile is used.

6 ISO Base Media File organization

6.1 Presentation structure

6.1.1 File Structure

A presentation may be contained in several files. One file contains the metadata for the whole presentation, and is formatted to this specification. This file may also contain all the media data, whereupon the presentation is self-contained. The other files, if used, are not required to be formatted to this specification; they are used to contain media data, and may also contain unused media data, or other information. This specification concerns the structure of the presentation file only. The format of the media-data files is constrained by this specification only in that the media-data in the media files must be capable of description by the metadata 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.

If an ISO file contains hint tracks, the media tracks that reference the media data from which the hints were built shall remain in the file, even if the data within them is not directly referenced by the hint tracks; after deleting all hint tracks, the entire un-hinted presentation shall remain. Note that the media tracks may, however, refer to external files for their media data.

Annex A provides an informative introduction, which may be of assistance to first-time readers.

6.1.2 Object Structure

The file is structured as a sequence of objects; some of these objects may contain other objects. The sequence of objects in the file shall contain exactly one presentation metadata wrapper (the Movie Box). It is usually close to the beginning or end of the file, to permit its easy location. The other objects found at this level may be a File-Type box, Free Space Boxes, Movie Fragments, Meta-data, or Media Data Boxes.

6.1.3 Meta Data and Media Data

The metadata is contained within the metadata wrapper (the Movie Box); the media data is contained either in the same file, within Media Data Box(es), or in other files. The media data is composed of images or audio data; the media data objects, or media data files, may contain other un-referenced information.

6.1.4 Track Identifiers

The track identifiers used in an ISO file are unique within that file; no two tracks shall use the same identifier.

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

6.2 Metadata Structure (Objects)

6.2.1 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 versions of this specification, 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 Track Reference Box): tmc, chap,
sync, scpt, ssr.
```

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).

6.2.2 Data Types and fields

In a number of boxes in this specification, 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), 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 optional in the specifications that use this specification. 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 version of this specification. 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 must 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 co-ordinates 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 co-ordinate 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 $\{a,b,u, c,d,v, x,y,w\}$.

6.2.3 Box Order

An overall view of the normal encapsulation structure is provided in the following table.

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 Track Header Box (tkhd) is found in a Track Box (trak), which is found in a Movie Box (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.

User data objects shall be placed only in Movie or Track Boxes, and 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 file type box 'ftyp' 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 Movie Header, Track Header, Media Header, and the specific media headers inside the Media Information Box (e.g. the Video Media Header).
- 3) Any Movie Fragment Boxes **shall** be in sequence order (see subclause 8.8.5).
- 4) It is **recommended** that the boxes within the Sample Table Box be in the following order: Sample Description, Time to Sample, Sample to Chunk, Sample Size, Chunk Offset.
- 5) It is strongly **recommended** that the Track Reference Box and Edit List (if any) **should** precede the Media Box, and the Handler Reference Box **should** precede the Media Information Box, and the Data Information Box **should** precede the Sample Table Box.
- 6) It is **recommended** that user Data Boxes be placed last in their container, which is either the Movie Box or Track Box.
- 7) It is **recommended** that the Movie Fragment Random Access Box, if present, be last in the file.

- 8) It is **recommended** that the progressive download information box be placed as early as possible in files, for maximum utility.

Table 1 — Box types, structure, and cross-reference

ftyp						*	4.3	file type and compatibility
pdin							8.1.3	progressive download information
moov						*	8.2.1	container for all the metadata
	mvhd					*	8.2.2	movie header, overall declarations
	trak					*	8.3.1	container for an individual track or stream
		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					8.6.4	edit list container
			elst				8.6.6	an edit list
		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
			minf			*	8.4.4	media information container
				vmhd			8.4.5.2	video media header, overall information (video track only)
				smhd			8.4.5.3	sound media header, overall information (sound track only)
				hmhd			8.4.5.4	hint media header, overall information (hint track only)
				nmhd			8.4.5.5	Null media header, overall information (some tracks only)
				dinf		*	8.5	data information box, container
					dref	*	8.7.2	data reference box, declares source(s) of media data in track
				stbl		*	8.5	sample table box, container for the time/space map
					stsd	*	8.5.2	sample descriptions (codec types, initialization etc.)
					stts	*	0	(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
	mvex						8.8.1	movie extends box
		mehd					8.8.2	movie extends header box
		trdx				*	8.8.3	track extends defaults
		leva					8.8.13	level assignment
moof							8.8.4	movie fragment
	mfhd					*	8.8.5	movie fragment header
	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

Table 1 (continued)

		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
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
	udta					8.10.1	user-data
		cppt				8.10.2	copyright etc.
		tset				8.10.3	track selection box
		strk				8.14.3	sub track box
			stri			8.14.4	sub track information box
			strd			8.14.5	sub track definition box
meta						8.11.1	metadata
	hdlr				*	8.4.3	handler, declares the metadata (handler) type
	dinf					8.5	data information box, container
		dref				8.7.2	data reference box, declares source(s) of metadata items
	iloc					8.11.3	item location
	ipro					8.11.5	item protection
		sinf				8.12.1	protection scheme information box
			frma			8.12.2	original format box
			schm			8.12.5	scheme type box
			schl			8.12.6	scheme information box
	iinf					8.11.6	item information
	xml					8.11.2	XML container
	bxml					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
meco						8.11.7	additional metadata container
	mere					8.11.8	metabox relation
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

6.2.4 URIs as type indicators

When URIs are used as a type indicator (e.g. in a sample entry or for un-timed meta-data), the URI must 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 must 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 Brand Identification

The definitions of the brands that apply to the file format are found in Annex E.

7 Streaming Support

7.1 Handling of Streaming Protocols

The file format supports streaming of media data over a network 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.

Transmission or server hint tracks contain instructions to assist a streaming server in the formation of packets for transmission. These instructions may contain immediate data for the server to send (e.g. header information) or reference segments of the media data. These instructions are encoded in the file in the same way that editing or presentation information is encoded in a file for local playback. Instead of editing or presentation information, information is provided which allows a server to packetize the media data in a manner suitable for streaming using a specific network transport.

The same media data is used in a file that contains hints, whether it is for local playback, or streaming over a number of different protocols. Separate 'hint' tracks for different protocols may be included within the same file and the media will play over all such protocols without making any additional copies of the media itself. In addition, existing media can be easily made streamable by the addition of appropriate hint tracks for specific protocols. The media data itself need not be recast or reformatted in any way.

This approach to streaming and recording is more space efficient than an approach that requires that the media information be partitioned into the actual data units that will be transmitted for a given transport and media format. Under such an approach, local playback requires either re-assembling the media from the packets, or having two copies of the media — one for local playback and one for streaming. Similarly, streaming such media over multiple protocols using this approach requires multiple copies of the media data for each transport. This is inefficient with space, unless the media data has been heavily transformed for streaming (e.g. by the application of error-correcting coding techniques, or by encryption).

Reception hint tracks may be used when one or more packet streams of data are recorded. Reception hint tracks indicate the order, reception timing, and contents of the received packets among other things.

NOTE Players may reproduce the packet stream that was received based on the reception hint tracks and process the reproduced packet stream as if it was newly received.

7.2 Protocol 'hint' tracks

Support for streaming is based upon the following three design parameters:

- The media data is represented as a set of network-independent standard tracks, which may be played, edited, and so on, as normal;
- There is a common declaration and base structure for hint tracks; this common format is protocol independent, but contains the declarations of which protocol(s) are described in the hint track(s);

- There is a specific design of the hint tracks for each protocol that may be transmitted; all these designs use the same basic structure. For example, there may be designs for RTP (for the Internet) and MPEG-2 transport (for broadcast), or for new standard or vendor-specific protocols.

The resulting streams, sent by the servers under the direction of the server hint tracks or reconstructed from the reception hint tracks, need contain no trace of file-specific information. This design does not require that the file structures or declaration style, be used either in the data on the wire or in the decoding station. For example, a file using ITU-T H.261 video and DVI audio, streamed under RTP, results in a packet stream that is fully compliant with the IETF specifications for packing those codings into RTP.

7.3 Hint Track Format

Hint tracks are used to describe elementary stream data in the file. Each protocol or each family of related protocols has its own hint track format. A server hint track format and a reception hint track format for the same protocol are distinguishable from the associated four-character code of the sample description entry. In other words, a different four-character code is used for a server hint track and a reception hint track of the same protocol. The syntax of the server hint track format and the reception hint track format for the same protocol should be the same or compatible so that a reception hint track can be used for re-sending of the stream provided that the potential degradations of the received streams are handled appropriately. Most protocols will need only one sample description format for each track.

Servers find their hint tracks by first finding all hint tracks, and then looking within that set for server hint tracks using their protocol (sample description format). If there are choices at this point, then the server chooses on the basis of preferred protocol or by comparing features in the hint track header or other protocol-specific information in the sample descriptions. Particularly in the absence of server hint tracks, servers may also use reception hint tracks of their protocol. However, servers should handle potential degradations of the received stream described by the used reception hint track appropriately.

Tracks having the `track_in_movie` flag set are candidates for playback, regardless of whether they are media tracks or reception hint tracks.

Hint tracks construct streams by pulling data out of other tracks by reference. These other tracks may be hint tracks or elementary stream tracks. The exact form of these pointers is defined by the sample format for the protocol, but in general they consist of four pieces of information: a track reference index, a sample number, an offset, and a length. Some of these may be implicit for a particular protocol. These 'pointers' always point to the actual source of the data. If a hint track is built 'on top' of another hint track, then the second hint track must have direct references to the media track(s) used by the first where data from those media tracks is placed in the stream.

All hint tracks use a common set of declarations and structures.

- Hint tracks are linked to the elementary stream tracks they carry, by track references of type `'hint'`
- They use a handler-type of `'hint'` in the Handler Reference Box
- They use a Hint Media Header Box
- They use a hint sample entry in the sample description, with a name and format unique to the protocol they represent.

Server hint tracks are usually marked as disabled for local playback, with their track header `track_in_movie` and `track_in_preview` flags set to 0.

Hint tracks may be created by an authoring tool, or may be added to an existing presentation by a hinting tool. Such a tool serves as a 'bridge' between the media and the protocol, since it intimately understands both. This permits authoring tools to understand the media format, but not protocols, and for servers to understand protocols (and their hint tracks) but not the details of media data.

Hint tracks do not use separate composition times; the 'ctts' table is not present in hint tracks. The process of hinting computes transmission times correctly as the decoding time.

NOTE 1: Servers using reception hint tracks as hints for sending of the received streams should handle the potential degradations of the received streams, such as transmission delay jitter and packet losses, gracefully and ensure that the constraints of the protocols and contained data formats are obeyed regardless of the potential degradations of the received streams.

NOTE 2: Conversion of received streams to media tracks allows existing players compliant with earlier versions of the ISO base media file format to process recorded files as long as the media formats are supported. However, most media coding standards only specify the decoding of error-free streams, and consequently it should be ensured that the content in media tracks can be correctly decoded. Players may utilize reception hint tracks for handling of degradations caused by the transmission, i.e., content that may not be correctly decoded is located only within reception hint tracks. The need for having a duplicate of the correct media samples in both a media track and a reception hint track can be avoided by including data from the media track by reference into the reception hint track.

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 Media Data Boxes. The actual media data follows the type field; its structure is described by the metadata (see particularly the sample table, subclause 8.5, and the item location box, subclause 8.11.3).

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 6.2, 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 metadata refers to media data by its absolute offset within the file (see subclause 8.7.5, the Chunk Offset Box); so Media Data Box 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 free-space box are irrelevant and may be ignored, or the object deleted, without affecting the presentation. (Care should be exercised when deleting the object, as this may invalidate the offsets used in the sample table, unless this object 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` may 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 Progressive download information box 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 progressive download information box 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.

8.2 Movie Structure

8.2.1 Movie Box

8.2.1.1 Definition

Box Type: 'moov'
 Container: File
 Mandatory: Yes
 Quantity: Exactly one

The metadata for a presentation is stored in the single Movie Box 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: Movie Box ('moov')
 Mandatory: Yes
 Quantity: Exactly one

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

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 specification)

`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 track identifier.

8.3 Track Structure

8.3.1 Track Box

8.3.1.1 Definition

Box Type: `'trak'`
 Container: Movie Box (`'moov'`)
 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 is independent of the other tracks in the presentation and carries its own temporal and spatial information. Each track will contain its associated Media Box.

Tracks are used for two purposes: (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 an ISO file, 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: Track Box ('trak')
 Mandatory: Yes
 Quantity: Exactly one

This box specifies the characteristics of a single track. Exactly one Track Header Box 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.

The default value of the track header flags for media tracks is 7 (track_enabled, track_in_movie, track_in_preview). If in a presentation all tracks have neither track_in_movie nor track_in_preview set, then all tracks shall be treated as if both flags were set on all tracks. Server hint tracks should have the track_in_movie and track_in_preview set to 0, so that they are ignored for local playback and preview.

Under the 'iso3' brand or brands that share its requirements, the width and height in the track header are measured on a notional 'square' (uniform) grid. Track video data is normalized to these dimensions (logically) before any transformation or placement caused by a layup or composition system. Track (and movie) matrices, if used, also operate in this uniformly-scaled space.

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 specification)

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

Track_enabled: Indicates that the track is enabled. Flag value is 0x000001. A disabled track (the low bit is zero) is treated as if it were not present.

Track_in_movie: Indicates that the track is used in the presentation. Flag value is 0x000002.

Track_in_preview: Indicates that the track is used when previewing the presentation. Flag value is 0x000004.

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 Movie Header Box). The value of this field is equal to the sum of the durations of all of the track's edits. If there is no edit list, then the duration is the sum of the sample durations, converted into the timescale in the Movie Header Box. If the duration of this track cannot be determined then duration is set to all 1s.

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 must 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 Movie Header Box 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** specify the track's visual presentation size as fixed-point 16.16 values. 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: Track Box ('trak')

Mandatory: No

Quantity: Zero or one

This box provides a reference from the containing track to another track in the presentation. These references are typed. A 'hint' reference links from the containing hint track to the media data that it hints. A content description reference 'cdsc' links a descriptive or metadata track to the content which it describes. The 'hind' dependency indicates that the referenced track(s) may contain media data required for decoding of the track containing the track reference. 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.

Exactly one Track Reference Box can be contained within the Track Box.

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') {
}

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

8.3.3.3 Semantics

The Track Reference Box contains track reference type boxes.

`track_ID` is an integer that provides a reference from the containing track to another track in the presentation. `track_IDs` are never re-used and cannot be equal to zero.

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' this track describes the referenced track.
- 'hind' this track depends on the referenced hint track, i.e., it should only be used if the referenced hint track is used.
- '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

8.3.4 Track Group Box

8.3.4.1 Definition

Box Type: 'trgr'
 Container: Track Box ('trak')
 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 Track Group Box 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 Track Reference Box is used for such purposes.

8.3.4.2 Syntax

```
aligned(8) class TrackGroupBox('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. The tracks that have the same value of `track_group_id` within a Group Type Box 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.

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

8.4 Track Media Structure

8.4.1 Media Box

8.4.1.1 Definition

Box Type: `'mdia'`
 Container: Track Box (`'trak'`)
 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: Media Box (`'mdia'`)
 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 time-scale for this media; 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 the duration of this media (in the scale of the timescale). If the duration cannot be determined then duration is set to all 1s.

language declares the language code for this media. See ISO 639-2/T for the set of three character codes. 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: Media Box ('mdia') or Meta Box ('meta')

Mandatory: Yes

Quantity: Exactly one

This box within a Media Box declares the process by which the media-data in the track is presented, and thus, the nature of the media in a track. For example, a video track would be handled by a video handler.

This box when present within a Meta Box, declares the structure or format of the 'meta' box 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. If they are in text, then a MIME format is supplied to document their format; if in XML, each sample is a complete XML document, and the namespace of the XML is also supplied.

An auxiliary video track is coded the same as a video track, but uses this different handler type, and is not intended to be visually displayed (e.g. it contains depth information, or other monochrome or color two-dimensional information). Auxiliary video tracks are usually linked to a video track by an appropriate track reference.

NOTE MPEG-7 streams, which are a specific kind of metadata stream, have their own handler declared, documented in the MP4 file format [ISO/IEC 14496-14].

NOTE metadata tracks are linked to the track they describe using a track-reference of type 'cdsc'. Metadata tracks use a null media header ('nmhd'), as defined in subclause 8.4.5.5.

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;
    string name;
}
```

8.4.3.3 Semantics

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

`handler_type` when present in a media box, is an integer containing one of the following values, or a value from a derived specification:

- 'vide' Video track
- 'soun' Audio track
- 'hint' Hint track
- 'meta' Timed Metadata track
- 'auxv' Auxiliary Video track

`handler_type` when present in a meta box, contains an appropriate value to indicate the format of the meta box contents. The value 'null' can be used in the primary meta box to indicate that it is merely being used to hold resources.

`name` is a null-terminated string in UTF-8 characters which 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: Media Box ('mdia')
 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

Box Types: 'vmhd', 'smhd', 'hmhd', 'nmhd'
 Container: Media Information Box ('minf')
 Mandatory: Yes
 Quantity: Exactly one specific media header shall be present

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 here, or one defined in a derived specification.

8.4.5.2 Video Media Header Box

The video media header contains general presentation information, independent of the coding, for video media. Note that the flags field has the value 1.

8.4.5.2.1 Syntax

```
aligned(8) class VideoMediaHeaderBox
    extends FullBox('vmhd', version = 0, 1) {
        template unsigned int(16) graphicsmode = 0;    // copy, see below
        template unsigned int(16)[3] opcolor = {0, 0, 0};
    }
```

8.4.5.2.2 Semantics

`version` is an integer that specifies the version of this box
`graphicsmode` specifies a composition mode for this video track, from the following enumerated set, which may be extended by derived specifications:
`copy = 0` copy over the existing image
`opcolor` is a set of 3 colour values (red, green, blue) available for use by graphics modes

8.4.5.3 Sound Media Header Box

The sound media header contains general presentation information, independent of the coding, for audio media. This header is used for all tracks containing audio.

8.4.5.3.1 Syntax

```
aligned(8) class SoundMediaHeaderBox
    extends FullBox('smhd', version = 0, 0) {
        template int(16) balance = 0;
        const unsigned int(16) reserved = 0;
    }
```

8.4.5.3.2 Semantics

`version` is an integer that specifies the version of this box
`balance` is a fixed-point 8.8 number that places mono audio tracks in a stereo space; 0 is centre (the normal value); full left is -1.0 and full right is 1.0.

8.4.5.4 Hint Media Header Box

The hint media header contains general information, independent of the protocol, for hint tracks. (A PDU is a Protocol Data Unit.)

8.4.5.4.1 Syntax

```
aligned(8) class HintMediaHeaderBox
    extends FullBox('hmhd', version = 0, 0) {
        unsigned int(16) maxPDUsize;
        unsigned int(16) avgPDUsize;
        unsigned int(32) maxbitrate;
        unsigned int(32) avgbitrate;
        unsigned int(32) reserved = 0;
    }
```

8.4.5.4.2 Semantics

`version` is an integer that specifies the version of this box
`maxPDUsize` gives the size in bytes of the largest PDU in this (hint) stream
`avgPDUsize` gives the average size of a PDU over the entire presentation
`maxbitrate` gives the maximum rate in bits/second over any window of one second
`avgbitrate` gives the average rate in bits/second over the entire presentation

8.4.5.5 Null Media Header Box

Streams other than visual and audio (e.g., timed metadata streams) may use a null Media Header Box, as defined here.

8.4.5.5.1 Syntax

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

8.4.5.5.2 Semantics

`version` - is an integer that specifies the version of this box.
`flags` - is a 24-bit integer with flags (currently all zero).

8.5 Sample Tables

8.5.1 Sample Table Box

8.5.1.1 Definition

Box Type: `'stbl'`
 Container: Media Information Box (`'minf'`)
 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 Sample Table Box references no data, then the Sample Table Box does not need to contain any sub-boxes (this is not a very useful media track).

If the track that the Sample Table Box is contained in does reference data, then the following sub-boxes are required: Sample Description, Sample Size, Sample To Chunk, and Chunk Offset. Further, the Sample Description Box shall contain at least one entry. A Sample Description Box is required because it contains the data reference index field which indicates which Data Reference Box to use to retrieve the media samples. Without the Sample Description, it is not possible to determine where the media samples are stored. The Sync Sample Box is optional. If the Sync Sample Box is not present, all samples are sync samples.

Annex A provides a narrative description of random access using the structures defined in the Sample Table Box.

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: 'stsd'

Container: Sample Table Box ('stbl')

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 information stored in the sample description box 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).

For video tracks, a VisualSampleEntry is used, for audio tracks, an AudioSampleEntry and for metadata tracks, a MetaDataSampleEntry. Hint tracks use an entry format specific to their protocol, with an appropriate name.

For hint tracks, the sample description contains appropriate declarative data for the streaming protocol being used, and the format of the hint track. The definition of the sample description is specific to the protocol.

Multiple descriptions may be used within a track.

The 'protocol' and 'codingname' fields are registered identifiers that uniquely identify the streaming protocol or compression format decoder to be used. A given protocol or codingname may have optional or required extensions to the sample description (e.g. codec initialization parameters). All such extensions shall be within boxes; these boxes occur after the required fields. Unrecognized boxes shall be ignored.

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

Note: 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.

The samplerate, samplesize and channelcount fields document the default audio output playback format for this media. The timescale for an audio track should be chosen to match the sampling rate, or be an integer multiple of it, to enable sample-accurate timing. ChannelCount is a value greater than zero that indicates the maximum number of channels that the audio could deliver. A ChannelCount of 1 indicates mono audio, and 2 indicates stereo (left/right). When values greater than 2 are used, the codec configuration should identify the channel assignment.

In video tracks, the frame_count field must be 1 unless the specification for the media format explicitly documents this template field and permits larger values. That specification must document both how the individual frames of video are found (their size information) and their timing established. That timing might be as simple as dividing the sample duration by the frame count to establish the frame duration.

NOTE 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

An optional BitRateBox may be present at the end of any MetaDataSampleEntry to signal the bit rate information of a stream. This can be used for buffer configuration. In case of XML metadata it can be used to choose the appropriate memory representation format (DOM, STX).

The width and height in the video sample entry document the pixel counts that the codec will deliver; this enables the allocation of buffers. Since these are counts they do not take into account pixel aspect ratio.

The pixel aspect ratio and clean aperture of the video may be specified using the 'pasp' and 'clap' sample entry boxes, respectively. These are both optional; if present, they over-ride the declarations (if any) in structures specific to the video codec, which structures should be examined if these boxes are absent. For maximum compatibility, these boxes should follow, not precede, any boxes defined in or required by derived specifications.

In the PixelAspectRatioBox, `hSpacing` and `vSpacing` have the same units, but those units are unspecified: only the ratio matters. `hSpacing` and `vSpacing` may or may not be in reduced terms, and they may reduce to 1/1. Both of them must be positive.

They are defined as the aspect ratio of a pixel, in arbitrary units. If a pixel appears H wide and V tall, then `hSpacing/vSpacing` is equal to H/V. This means that a square on the display that is n pixels tall needs to be $n \times vSpacing/hSpacing$ pixels wide to appear square.

NOTE When adjusting pixel aspect ratio, normally, the horizontal dimension of the video is scaled, if needed (i.e. if the final display system has a different pixel aspect ratio from the video source).

NOTE It is recommended that the original pixels, and the composed transform, be carried through the pipeline as far as possible. If the transformation resulting from 'correcting' pixel aspect ratio to a square grid, normalizing to the track dimensions, composition or placement (e.g. track and/or movie matrix), and normalizing to the display characteristics, is a unity matrix, then no re-sampling need be done. In particular, video should not be re-sampled more than once in the process of rendering, if at all possible.

There are notionally four values in the CleanApertureBox. These parameters are represented as a fraction N/D. The fraction may or may not be in reduced terms. We refer to the pair of parameters `fooN` and `fooD` as `foo`. For `horizOff` and `vertOff`, D must be positive and N may be positive or negative. For `cleanApertureWidth` and `cleanApertureHeight`, both N and D must be positive.

NOTE These are fractional numbers for several reasons. First, in some systems the exact width after pixel aspect ratio correction is integral, not the pixel count before that correction. Second, if video is resized in the full aperture, the exact expression for the clean aperture may not be integral. Finally, because this is represented using centre and offset, a division by two is needed, and so half-values can occur.

Considering the pixel dimensions as defined by the VisualSampleEntry width and height. If picture centre of the image is at `pcX` and `pcY`, then `horizOff` and `vertOff` are defined as follows:

```
pcX = horizOff + (width - 1)/2
pcY = vertOff + (height - 1)/2;
```

Typically, `horizOff` and `vertOff` are zero, so the image is centred about the picture centre.

The leftmost/rightmost pixel and the topmost/bottommost line of the clean aperture fall at:

```
pcX ± (cleanApertureWidth - 1)/2
pcY ± (cleanApertureHeight - 1)/2;
```

The audio output format (samplerate, samplesize and channelcount fields) in the sample entry should be considered definitive only for codecs that do not record their own output configuration. If the audio codec has definitive information about the output format, it shall be taken as definitive; in this case the samplerate, samplesize and channelcount fields in the sample entry may be ignored, though sensible values should be chosen (for example, the highest possible sampling rate).

The URIMetaSampleEntry entry contains, in a box, the URI defining the form of the metadata, and optional initialization data. The format of both the samples and of the initialization data is defined by all or part of the URI form.

An optional bitrate box may be used in the URIMetaSampleEntry entry, as usual.

It may be the case that the URI identifies a format of metadata that allows there to be more than one 'stated fact' within each sample. However, all metadata samples in this format are effectively 'I frames', defining the entire set of metadata for the time interval they cover. This means that the complete set of metadata at any

instant, for a given track, is contained in (a) the time-aligned samples of the track(s) (if any) describing that track, plus (b) the track metadata (if any), the movie metadata (if any) and the file metadata (if any).

If incrementally-changed metadata is needed, the MPEG-7 framework provides that capability.

Information on URI forms for some metadata systems can be found in Annex G.

Colour information may be supplied in one or more ColourInformationBoxes placed in a VisualSampleEntry. These should be placed in order in the sample entry starting with the most accurate (and potentially the most difficult to process), in progression to the least. These are advisory and concern rendering and colour conversion, and there is no normative behaviour associated with them; a reader may choose to use the most suitable. A ColourInformationBox with an unknown colour type may be ignored.

If used, an ICC profile may be a restricted one, under the code 'rICC', which permits simpler processing. That profile shall be of either the Monochrome or Three-Component Matrix-Based class of input profiles, as defined by ISO 15076-1. If the profile is of another class, then the 'prof' indicator must be used.

If colour information is supplied in both this box, and also in the video bitstream, this box takes precedence, and over-rides the information in the bitstream.

NOTE When an ICC profile is specified, SMPTE RP 177 "Derivation of Basic Television Color Equations" may be of assistance if there is a need to form the Y'CbCr to R'G'B' conversion matrix for the color primaries described by the ICC profile.

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 HintSampleEntry() extends SampleEntry (protocol) {
    unsigned int(8) data [];
}

class BitRateBox extends Box('btrt'){
    unsigned int(32) bufferSizeDB;
    unsigned int(32) maxBitrate;
    unsigned int(32) avgBitrate;
}

class MetaDataSampleEntry(codingname) extends SampleEntry (codingname) {
}

class XMLMetaDataSampleEntry() extends MetaDataSampleEntry ('metx') {
    string    content_encoding; // optional
    string    namespace;
    string    schema_location; // optional
    BitRateBox (); // optional
}

class TextMetaDataSampleEntry() extends MetaDataSampleEntry ('mett') {
    string    content_encoding; // optional
    string    mime_format;
    BitRateBox (); // optional
}
```

```

aligned(8) class URIBox
    extends FullBox('uri ', version = 0, 0) {
    string    theURI;
}

aligned(8) class URIInitBox
    extends FullBox('uriI', version = 0, 0) {
    unsigned int(8) uri_initialization_data[];
}

class URIMetaSampleEntry() extends MetaDataSampleEntry ('urim') {
    URIbox        the_label;
    URIInitBox    init;      // optional
    MPEG4BitRateBox ();      // optional
}

// Visual Sequences

class PixelAspectRatioBox extends Box('pasp'){
    unsigned int(32) hSpacing;
    unsigned int(32) vSpacing;
}

class CleanApertureBox extends Box('clap'){
    unsigned int(32) cleanApertureWidthN;
    unsigned int(32) cleanApertureWidthD;

    unsigned int(32) cleanApertureHeightN;
    unsigned int(32) cleanApertureHeightD;

    unsigned int(32) horizOffN;
    unsigned int(32) horizOffD;

    unsigned int(32) vertOffN;
    unsigned int(32) vertOffD;
}

class ColourInformationBox extends Box('colr'){
    unsigned int(32) colour_type;
    if (colour_type == 'nclx') /* on-screen colours */
    {
        unsigned int(16) colour_primaries;
        unsigned int(16) transfer_characteristics;
        unsigned int(16) matrix_coefficients;
        unsigned int(1)  full_range_flag;
        unsigned int(7)  reserved = 0;
    }
    else if (colour_type == 'rICC')
    {
        ICC_profile;    // restricted ICC profile
    }
    else if (colour_type == 'prof')
    {
        ICC_profile;    // unrestricted ICC profile
    }
}

class VisualSampleEntry(codingname) extends SampleEntry (codingname){
    unsigned int(16) pre_defined = 0;
    const unsigned int(16) reserved = 0;
    unsigned int(32)[3] pre_defined = 0;
    unsigned int(16) width;
    unsigned int(16) height;
}

```

```

template unsigned int(32)  horizresolution = 0x00480000; // 72 dpi
template unsigned int(32)  vertresolution  = 0x00480000; // 72 dpi
const unsigned int(32)  reserved = 0;
template unsigned int(16)  frame_count = 1;
string[32]  compressorname;
template unsigned int(16)  depth = 0x0018;
int(16)  pre_defined = -1;
// other boxes from derived specifications
CleanApertureBox      clap; // optional
PixelAspectRatioBox  pasp;   // optional
}

// Audio Sequences

class AudioSampleEntry(codingname) extends SampleEntry (codingname){
    const unsigned int(32)[2] reserved = 0;
    template unsigned int(16) channelcount = 2;
    template unsigned int(16) samplesize = 16;
    unsigned int(16) pre_defined = 0;
    const unsigned int(16) reserved = 0 ;
    template unsigned int(32) samplerate = { default samplerate of media}<<16;
}

aligned(8) class SampleDescriptionBox (unsigned int(32) handler_type)
    extends FullBox('stsd', 0, 0){
    int i ;
    unsigned int(32) entry_count;
    for (i = 1 ; i <= entry_count ; i++){
        switch (handler_type){
            case 'soun': // for audio tracks
                AudioSampleEntry();
                break;
            case 'vide': // for video tracks
                VisualSampleEntry();
                break;
            case 'hint': // Hint track
                HintSampleEntry();
                break;
            case 'meta': // Metadata track
                MetadataSampleEntry();
                break;
        }
    }
}
}

```

8.5.2.3 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

SampleEntry is the appropriate sample entry.

data_reference_index is an integer that contains the index of the data reference to use to retrieve data associated with samples that use this sample description. Data references are stored in Data Reference Boxes. The index ranges from 1 to the number of data references.

ChannelCount is the number of channels such as 1 (mono) or 2 (stereo)

`SampleSize` is in bits, and takes the default value of 16
`SampleRate` is the sampling rate expressed as a 16.16 fixed-point number (hi.lo)
`resolution` fields give the resolution of the image in pixels-per-inch, as a fixed 16.16 number
`frame_count` indicates how many frames of compressed video are stored in each sample. The default is 1, for one frame per sample; it may be more than 1 for multiple frames per sample
`Compressorname` is a name, for informative purposes. It is formatted in a fixed 32-byte field, with the first byte set to the number of bytes to be displayed, followed by that number of bytes of displayable data, and then padding to complete 32 bytes total (including the size byte). The field may be set to 0.
`depth` takes one of the following values
0x0018 – images are in colour with no alpha
`width` and `height` are the maximum visual width and height of the stream described by this sample description, in pixels
`hSpacing`, `vSpacing`: define the relative width and height of a pixel;
`cleanApertureWidthN`, `cleanApertureWidthD`: a fractional number which defines the exact clean aperture width, in counted pixels, of the video image
`cleanApertureHeightN`, `cleanApertureHeightD`: a fractional number which defines the exact clean aperture height, in counted pixels, of the video image
`horizOffN`, `horizOffD`: a fractional number which defines the horizontal offset of clean aperture centre minus (width-1)/2. Typically 0.
`vertOffN`, `vertOffD`: a fractional number which defines the vertical offset of clean aperture centre minus (height-1)/2. Typically 0.
`content_encoding` - is a null-terminated string in UTF-8 characters, and provides a MIME type which identifies the content encoding of the timed metadata. It is defined in the same way as for an `ItemInfoEntry` in this specification. If not present (an empty string is supplied) the timed metadata is not encoded. An example for this field is 'application/zip'. Note that no MIME types for BiM [ISO/IEC 23001-1] and TeM [ISO/IEC 15938-1] currently exist. Thus the experimental MIME types 'application/x-BiM' and 'text/x-TeM' shall be used to identify these encoding mechanisms.
`namespace` - gives the namespace of the schema for the timed XML metadata. This is needed for identifying the type of metadata, e.g. gBSD or AQoS [MPEG-21-7] and for decoding using XML aware encoding mechanisms such as BiM.
`schema_location` - optionally provides an URL to find the schema corresponding to the namespace. This is needed for decoding of the timed metadata by XML aware encoding mechanisms such as BiM.
`mime_format` - provides a MIME type which identifies the content format of the timed metadata. Examples for this field are 'text/html' and 'text/plain'.
`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.
`avgBitrate` gives the average rate in bits/second over the entire presentation.
`theURI` is a URI formatted according to the rules in 6.2.4;
`uri_initialization_data` is opaque data whose form is defined in the documentation of the URI form.
`colour_type`: an indication of the type of colour information supplied. For `colour_type` 'nclx': these fields are exactly the four bytes defined for `PTM_COLOR_INFO( )` in A.7.2 of ISO/IEC 29199-2 but note that the full range flag is here in a different bit position
`ICC_profile`: an ICC profile as defined in ISO 15076-1 or ICC.1:2010 is supplied.

8.5.3 Degradation Priority Box

8.5.3.1 Definition

Box Type: 'stdp'
Container: Sample Table Box ('stbl').
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 Sample Size Box ('stsz'). 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

(empty sub-clause)

8.6 Track Time Structures

8.6.1 Time to Sample Boxes

8.6.1.1 Definition

The composition times (CT) and decoding times (DT) of samples are derived from the Time to Sample Boxes, of which there are two types. The decoding time is defined in the Decoding Time to Sample Box, giving time deltas between successive decoding times. The composition times are derived in the Composition Time to Sample Box as composition time offsets from decoding time. If the composition times and decoding times are identical for every sample in the track, then only the Decoding Time to Sample Box is required; the composition time to sample box must not be present.

The time to sample boxes must give non-zero durations for all samples with the possible exception of the last one. Durations in the `'stts'` box are strictly positive (non-zero), except for the very last entry, which may be zero. This rule derives from the rule that no two time-stamps in a stream may be the same. 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. If the duration of the last sample is indeterminate, use an arbitrary small value and a 'dwell' edit.

In the following example, there is a sequence of I, P, and B frames, each with a decoding time delta of 10. The samples are stored as follows, with the indicated values for their decoding time deltas and composition time offsets (the actual CT and 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 deltas of the preceding samples. Note that the total of the decoding deltas 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
Decode delta	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
Decode Delta	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: Sample Table Box ('stbl')
 Mandatory: Yes
 Quantity: Exactly one

This box contains a compact version of a table that allows indexing from decoding time 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 time delta, and the delta of those samples. By adding the deltas a complete time-to-sample map may be built.

The Decoding Time to Sample Box contains decode time delta's: $DT(n+1) = DT(n) + STTS(n)$ where $STTS(n)$ is the (uncompressed) table entry for sample n .

The sample entries are ordered by decoding time stamps; therefore the deltas are all non-negative.

The DT axis has a zero origin; $DT(i) = \text{SUM}(\text{for } j=0 \text{ to } i-1 \text{ of } \text{delta}(j))$, and the sum of all deltas gives the length of the media in the track (not mapped to the overall timescale, and not considering any edit list).

The Edit List Box provides the initial CT value if it is non-empty (non-zero).

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 delta of these samples 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: Sample Table Box (`'stbl'`)

Mandatory: No

Quantity: Zero or one

This box provides the offset between decoding time and composition time. In version 0 of this box the decoding time must be less than the composition time, and the offsets are expressed as unsigned numbers such that $CT(n) = DT(n) + CTTS(n)$ where $CTTS(n)$ is the (uncompressed) table entry for sample n . In version 1 of this box, the composition timeline and the decoding timeline are still derived from each other, but the offsets are signed. It is recommended that for the computed composition timestamps, there is exactly one with the value 0 (zero).

For either version of the box, each sample must have a unique composition timestamp value, that is, the timestamp for two samples shall never be the same.

It may be true that there is no frame to compose at time 0; the handling of this is unspecified (systems might display the first frame for longer, or a suitable fill colour).

When version 1 of this box is used, the CompositionToDecodeBox may also be present in the sample table to relate the composition and decoding timelines. When backwards-compatibility or compatibility with an unknown set of readers is desired, version 0 of this box should be used when possible. In either version of this box, but particularly under version 0, if it is desired that the media start at track time 0, and the first media sample does not have a composition time of 0, an edit list may be used to 'shift' the media to time 0.

The composition time to sample table is optional and must only be present if DT and CT differ for any samples.

Hint tracks do not use this box.

For example in Table 2

Sample count	Sample_offset
1	10
1	30
2	0
1	30
2	0
1	10
1	30
2	0
1	30
2	0

8.6.1.3.2 Syntax

```
aligned(8) class CompositionOffsetBox
    extends FullBox('ctts', version = 0, 0) {
    unsigned int(32) entry_count;
    int i;
    if (version==0) {
        for (i=0; i < entry_count; i++) {
            unsigned int(32) sample_count;
            unsigned int(32) sample_offset;
        }
    }
    else if (version == 1) {
        for (i=0; i < entry_count; i++) {
            unsigned int(32) sample_count;
            signed int(32) sample_offset;
        }
    }
}
```

8.6.1.3.3 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 offset.

sample_offset is an integer that gives the offset between CT and DT, such that $CT(n) = DT(n) + CTTS(n)$.

8.6.1.4 Composition to Decode Box

8.6.1.4.1 Definition

Box Type: 'cslg'

Container: Sample Table Box ('stbl')

Mandatory: No

Quantity: Zero or one

When signed composition offsets are used, this box may be used to relate the composition and decoding timelines, and deal with some of the ambiguities that signed composition offsets introduce.

Note that all these fields apply to the entire media (not just that selected by any edits). It is recommended that any edits, explicit or implied, not select any portion of the composition timeline that does not map to a sample. For example, if the smallest composition time is 1000, then the default edit from 0 to the media duration leaves the period from 0 to 1000 associated with no media sample. Player behaviour, and what is composed in this interval, is undefined under these circumstances. It is recommended that the smallest computed CTS be zero, or match the beginning of the first edit.

The composition duration of the last sample in a track might be (often is) ambiguous or unclear; the field for composition end time can be used to clarify this ambiguity and, with the composition start time, establish a clear composition duration for the track.

When the Composition to Decode Box is included in the Sample Table Box, it documents the composition and decoding time relations of the samples in the Movie Box only, not including any subsequent movie fragments.

8.6.1.4.2 Syntax

```
class CompositionToDecodeBox extends FullBox('cslg', version=0, 0) {
    signed int(32) compositionToDTSShift;
    signed int(32) leastDecodeToDisplayDelta;
    signed int(32) greatestDecodeToDisplayDelta;
    signed int(32) compositionStartTime;
    signed int(32) compositionEndTime;
}
```

8.6.1.4.3 Semantics

compositionToDTSShift: if this value is added to the composition times (as calculated by the CTS offsets from the DTS), then for all samples, their CTS is guaranteed to be greater than or equal to their DTS, and the buffer model implied by the indicated profile/level will be honoured; if **leastDecodeToDisplayDelta** is positive or zero, this field can be 0; otherwise it should be at least $(- \text{leastDecodeToDisplayDelta})$

leastDecodeToDisplayDelta: the smallest composition offset in the CompositionTimeToSample box in this track

greatestDecodeToDisplayDelta: the largest composition offset in the CompositionTimeToSample box in this track

compositionStartTime: the smallest computed composition time (CTS) for any sample in the media of this track

compositionEndTime: the composition time plus the composition duration, of the sample with the largest computed composition time (CTS) in the media of this track; if this field takes the value 0, the composition end time is unknown.

8.6.2 Sync Sample Box

8.6.2.1 Definition

Box Type: 'stss'
 Container: Sample Table Box ('stbl')
 Mandatory: No
 Quantity: Zero or one

This box provides a compact marking of the sync samples within the stream. The table is arranged in strictly increasing order of sample number.

If the sync sample box is not present, every sample is a sync sample.

8.6.2.2 Syntax

```
aligned(8) class SyncSampleBox
    extends FullBox('stss', version = 0, 0) {
    unsigned int(32) entry_count;
    int i;
    for (i=0; i < entry_count; i++) {
        unsigned int(32) sample_number;
    }
}
```

8.6.2.3 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. If **entry_count** is zero, there are no sync samples within the stream and the following table is empty.

sample_number gives the numbers of the samples that are sync samples in the stream.