INTERNATIONAL STANDARD

ISO/IEC 14496-12

Fourth edition
2012-07-15

Corrected version
2012-09-15

Information technology — Coding of audio-visual objects —

Part 12: ISO base media file format

Technologies de l'information — Codage des objets audiovisuels —

Partie 12: Format ISO de base pour les fichiers médias

Reference number
ISO/IEC 14496-12:2012(E)

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

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 14496-12 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 14496-12:2008) of which it constitutes a minor revision. It also incorporates the Amendment ISO/IEC 14496-12:2008/Amd.1:2009 and the Technical Corrigenda ISO/IEC 14496-12:2008/Cor.1:2008, ISO/IEC 14496-12:2008/Cor.2:2009, ISO/IEC 14496-12:2008/Cor.3:2009, and ISO/IEC 14496-12:2008/Cor.4:2011.

ISO/IEC 14496 consists of the following parts, under the general title *Information technology — Coding of audio-visual objects*:

— *Part 1: Systems*

— *Part 2: Visual*

— *Part 3: Audio*

— *Part 4: Conformance testing*

— *Part 5: Reference software*

— *Part 6: Delivery Multimedia Integration Framework (DMIF)*

— *Part 7: Optimized reference software for coding of audio-visual objects* [Technical Report]

— *Part 8: Carriage of ISO/IEC 14496 contents over IP networks*

— *Part 9: Reference hardware description* [Technical Report]

— *Part 10: Advanced Video Coding*

— *Part 11: Scene description and application engine*

— *Part 12: ISO base media file format*

— *Part 13: Intellectual Property Management and Protection (IPMP) extensions*

— *Part 14: MP4 file format*

— *Part 15: Advanced Video Coding (AVC) file format*

— *Part 16: Animation Framework eXtension (AFX)*

— *Part 17: Streaming text format*

— *Part 18: Font compression and streaming*

— *Part 19: Synthesized texture stream*

— *Part 20: Lightweight Application Scene Representation (LASeR) and Simple Aggregation Format (SAF)*

— *Part 21: MPEG-J Graphics Framework eXtensions (GFX)*

— *Part 22: Open Font Format*

— *Part 23: Symbolic Music Representation*

— *Part 24: Audio and systems interaction* [Technical Report]

— *Part 25: 3D Graphics Compression Model*

— *Part 26: Audio conformance*

— *Part 27: 3D Graphics conformance*

— *Part 28: Composite font representation*

This corrected version of ISO/IEC 14496-12:2012 incorporates the corrections made by ISO/IEC 14496-12:2008 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 — Coding of audio-visual objects —

Part 12:
ISO base media file format

1 Scope

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

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.

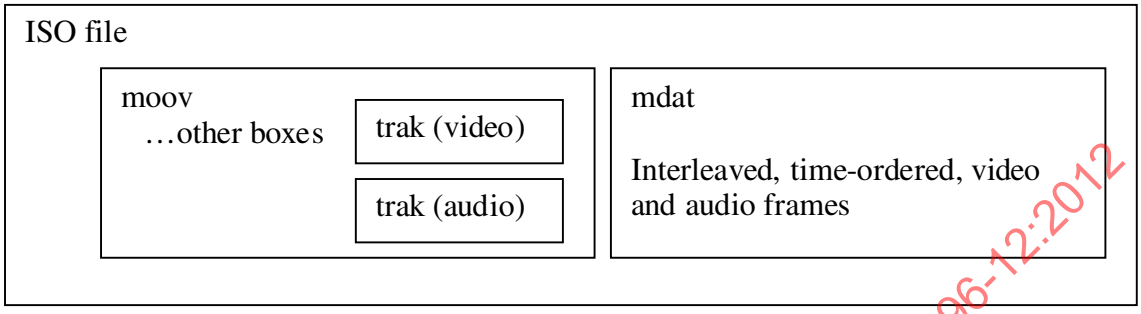

Figure 1 — Simple interchange file

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.

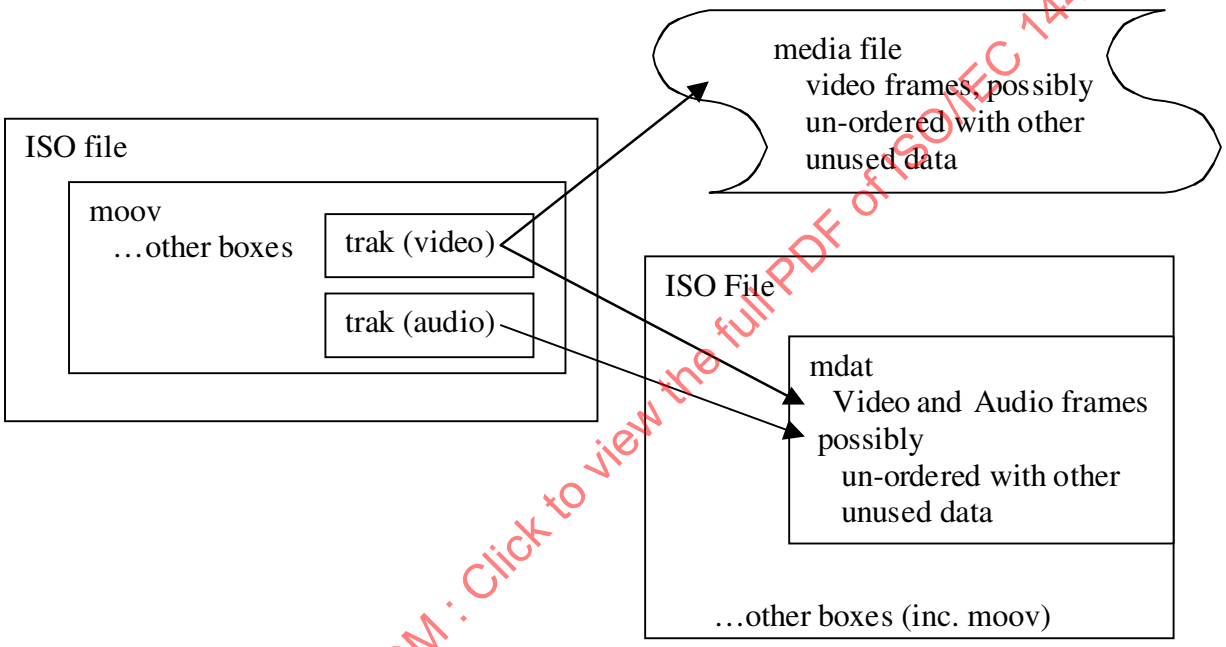

Figure 2 — Content Creation File

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.

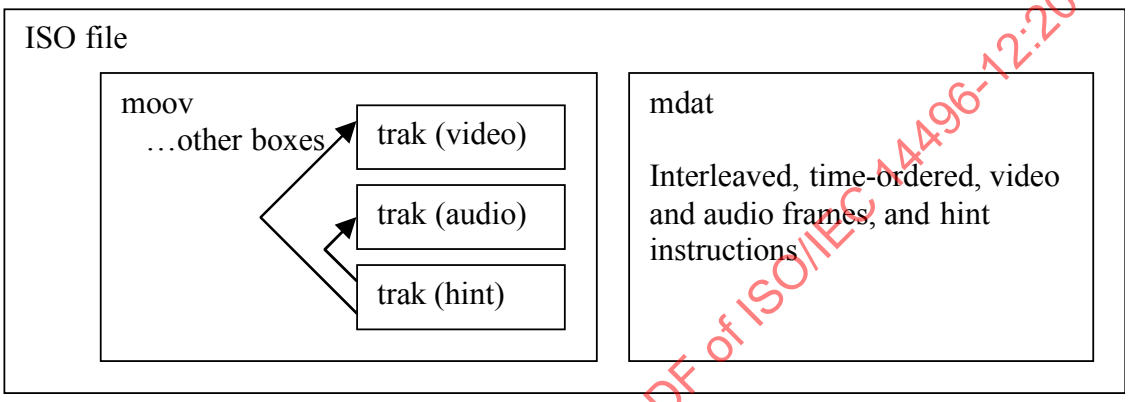

Figure 3 — Hinted Presentation for Streaming

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

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

6.2.2 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:

```
(p q 1) *    | a   b   u |   = (m n z)
             | c   d   v |
             | x   y   w |

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

p' = m/z;   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*

Table 1 (*continued*)

			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	*	8.6.1.2	*(decoding) time-to-sample*
					ctts		8.6.1.3	*(composition) time to sample*
					cslg		8.6.1.4	*composition to decode timeline mapping*
					stsc	*	8.7.4	*sample-to-chunk, partial data-offset information*
					stsz		8.7.3.2	*sample sizes (framing)*
					stz2		8.7.3.3	*compact sample sizes (framing)*
					stco	*	8.7.5	*chunk offset, partial data-offset information*
					co64		8.7.5	*64-bit chunk offset*
					stss		8.6.2	*sync sample table*
					stsh		8.6.3	*shadow sync sample table*
					padb		8.7.6	*sample padding bits*
					stdp		8.7.6	*sample degradation priority*
					sdtp		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*
		trex				*	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*
		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*
		cprt					8.10.2	*copyright etc.*
		tsel					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*

Table 1 (*continued*)

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*
			schi				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 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 4.2, is used.

There may be any number of these boxes in the file (including zero, if all the media data is in other files). The 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.

A.7 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*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) = SUM(for j=0 to i-1 of 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 `(- 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.

8.6.3 Shadow Sync Sample Box

8.6.3.1 Definition

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

The shadow sync table provides an optional set of sync samples that can be used when seeking or for similar purposes. In normal forward play they are ignored.

Each entry in the ShadowSyncTable consists of a pair of sample numbers. The first entry (shadowed-sample-number) indicates the number of the sample that a shadow sync will be defined for. This should always be a non-sync sample (e.g. a frame difference). The second sample number (sync-sample-number) indicates the sample number of the sync sample (i.e. key frame) that can be used when there is a need for a sync sample at, or before, the shadowed-sample-number.

The entries in the ShadowSyncBox shall be sorted based on the shadowed-sample-number field.

The shadow sync samples are normally placed in an area of the track that is not presented during normal play (edited out by means of an edit list), though this is not a requirement. The shadow sync table can be ignored and the track will play (and seek) correctly if it is ignored (though perhaps not optimally).

The ShadowSyncSample replaces, not augments, the sample that it shadows (i.e. the next sample sent is shadowed-sample-number+1). The shadow sync sample is treated as if it occurred at the time of the sample it shadows, having the duration of the sample it shadows.

Hinting and transmission might become more complex if a shadow sample is used also as part of normal playback, or is used more than once as a shadow. In this case the hint track might need separate shadow syncs, all of which can get their media data from the one shadow sync in the media track, to allow for the different time-stamps etc. needed in their headers.

8.6.3.2 Syntax

```
aligned(8) class ShadowSyncSampleBox
   extends FullBox('stsh', version = 0, 0) {
   unsigned int(32)  entry_count;
   int i;
   for (i=0; i < entry_count; i++) {
      unsigned int(32)  shadowed_sample_number;
      unsigned int(32)  sync_sample_number;
   }
}
```

8.6.3.3 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.
`shadowed_sample_number` - gives the number of a sample for which there is an alternative sync sample.
`sync_sample_number` - gives the number of the alternative sync sample.

8.6.4 Independent and Disposable Samples Box

8.6.4.1 Definition

Box Types: 'sdtp'
Container: Sample Table Box ('stbl')
Mandatory: No
Quantity: Zero or one

This optional table answers three questions about sample dependency:

1) does this sample depend on others (e.g. is it an I-picture)?
2) do no other samples depend on this one?
3) does this sample contain multiple (redundant) encodings of the data at this time-instant (possibly with different dependencies)?

In the absence of this table:

1) the sync sample table (partly) answers the first question; in most video codecs, I-pictures are also sync points,
2) the dependency of other samples on this one is unknown.
3) the existence of redundant coding is unknown.

When performing 'trick' modes, such as fast-forward, it is possible to use the first piece of information to locate independently decodable samples. Similarly, when performing random access, it may be necessary to locate the previous sync sample or random access recovery point, and roll-forward from the sync sample or the pre-roll starting point of the random access recovery point to the desired point. While rolling forward, samples on which no others depend need not be retrieved or decoded.

The value of 'sample_is_depended_on' is independent of the existence of redundant codings. However, a redundant coding may have different dependencies from the primary coding; if redundant codings are available, the value of 'sample_depends_on' documents only the primary coding.

A leading sample (usually a picture in video) is defined relative to a reference sample, which is the immediately prior sample that is marked as "sample_depends_on" having no dependency (an I picture). A leading sample has both a composition time before the reference sample, and possibly also a decoding dependency on a sample before the reference sample. Therefore if, for example, playback and decoding were to start at the reference sample, those samples marked as leading would not be needed and might not be decodable. A leading sample itself must therefore not be marked as having no dependency.

For tracks with a `handler_type` that is not 'vide', 'soun', 'hint' or 'auxv', if another sample with `sample_depends_on=2` or another sample tagged as a "Sync Sample" has already been processed and unless specified otherwise, a sample tagged with `sample_depends_on=2`, and `sample_has_redundancy=1` can be discarded, and its duration added to the duration of the preceding one, to maintain the timing of subsequent samples.

The size of the table, `sample_count`, is taken from the `sample_count` in the Sample Size Box (`'stsz'`) or Compact Sample Size Box (`'stz2'`).

8.6.4.2 Syntax

```
aligned(8) class SampleDependencyTypeBox
   extends FullBox('sdtp', version = 0, 0) {
   for (i=0; i < sample_count; i++){
      unsigned int(2) is_leading;
      unsigned int(2) sample_depends_on;
      unsigned int(2) sample_is_depended_on;
      unsigned int(2) sample_has_redundancy;
   }
}
```

8.6.4.3 Semantics

`is_leading` takes one of the following four values:
- 0: the leading nature of this sample is unknown;
- 1: this sample is a leading sample that has a dependency before the referenced I-picture (and is therefore not decodable);
- 2: this sample is not a leading sample;
- 3: this sample is a leading sample that has no dependency before the referenced I-picture (and is therefore decodable);

`sample_depends_on` takes one of the following four values:
- 0: the dependency of this sample is unknown;
- 1: this sample does depend on others (not an I picture);
- 2: this sample does not depend on others (I picture);
- 3: reserved

`sample_is_depended_on` takes one of the following four values:
- 0: the dependency of other samples on this sample is unknown;
- 1: other samples may depend on this one (not disposable);
- 2: no other sample depends on this one (disposable);
- 3: reserved

`sample_has_redundancy` takes one of the following four values:
- 0: it is unknown whether there is redundant coding in this sample;
- 1: there is redundant coding in this sample;
- 2: there is no redundant coding in this sample;
- 3: reserved

8.6.5 Edit Box

8.6.5.1 Definition

Box Type: `'edts'`
Container: Track Box (`'trak'`)
Mandatory: No
Quantity: Zero or one

An Edit Box maps the presentation time-line to the media time-line as it is stored in the file. The Edit Box is a container for the edit lists.

The Edit Box is optional. In the absence of this box, there is an implicit one-to-one mapping of these time-lines, and the presentation of a track starts at the beginning of the presentation. An empty edit is used to offset the start time of a track.

8.6.5.2 Syntax

```
aligned(8) class EditBox extends Box('edts') {
}
```

8.6.6 Edit List Box

8.6.6.1 Definition

Box Type: `'elst'`
Container: Edit Box (`'edts'`)
Mandatory: No
Quantity: Zero or one

This box contains an explicit timeline map. Each entry defines part of the track time-line: by mapping part of the media time-line, or by indicating 'empty' time, or by defining a 'dwell', where a single time-point in the media is held for a period.

NOTE Edits are not restricted to fall on sample times. This means that when entering an edit, it can be necessary to (a) back up to a sync point, and pre-roll from there and then (b) be careful about the duration of the first sample — it might have been truncated if the edit enters it during its normal duration. If this is audio, that frame might need to be decoded, and then the final slicing done. Likewise, the duration of the last sample in an edit might need slicing.

Starting offsets for tracks (streams) are represented by an initial empty edit. For example, to play a track from its start for 30 seconds, but at 10 seconds into the presentation, we have the following edit list:

Entry-count = 2

Segment-duration = 10 seconds
Media-Time = -1
Media-Rate = 1

Segment-duration = 30 seconds (could be the length of the whole track)
Media-Time = 0 seconds
Media-Rate = 1

A non-empty edit may insert a portion of the media timeline that is not present in the initial movie, and is present only in subsequent movie fragments. Particularly in an empty initial movie of a fragmented movie file (when there are no media samples yet present), the `segment_duration` of this edit may be zero, whereupon the edit provides the offset from media composition time to movie presentation time, for the movie and subsequent movie fragments. It is recommended that such an edit be used to establish a presentation time of 0 for the first presented sample, when composition offsets are used.

For example, if the composition time of the first composed frame is 20, then the edit that maps the media time from 20 onwards to movie time 0 onwards, would read:

Entry-count = 1

Segment-duration = 0
Media-Time = 20
Media-Rate = 1

8.6.6.2 Syntax

```
aligned(8) class EditListBox extends FullBox('elst', version, 0) {
   unsigned int(32)  entry_count;
   for (i=1; i <= entry_count; i++) {
      if (version==1) {
         unsigned int(64) segment_duration;
         int(64) media_time;
      } else { // version==0
         unsigned int(32) segment_duration;
         int(32)  media_time;
      }
      int(16) media_rate_integer;
      int(16) media_rate_fraction = 0;
   }
}
```

8.6.6.3 Semantics

`version` is an integer that specifies the version of this box (0 or 1)
`entry_count` is an integer that gives the number of entries in the following table
`segment_duration` is an integer that specifies the duration of this edit segment in units of the timescale in the Movie Header Box
`media_time` is an integer containing the starting time within the media of this edit segment (in media time scale units, in composition time). If this field is set to –1, it is an empty edit. The last edit in a track shall never be an empty edit. Any difference between the duration in the Movie Header Box, and the track's duration is expressed as an implicit empty edit at the end.
`media_rate` specifies the relative rate at which to play the media corresponding to this edit segment. If this value is 0, then the edit is specifying a 'dwell': the media at media-time is presented for the segment-duration. Otherwise this field shall contain the value 1.

8.7 Track Data Layout Structures

8.7.1 Data Information Box

8.7.1.1 Definition

Box Type: `'dinf'`
Container: Media Information Box (`'minf'`) or Meta Box (`'meta'`)
Mandatory: Yes (required within `'minf'` box) and No (optional within `'meta'` box)
Quantity: Exactly one

The data information box contains objects that declare the location of the media information in a track.

8.7.1.2 Syntax

```
aligned(8) class DataInformationBox extends Box('dinf') {
}
```

8.7.2 Data Reference Box

8.7.2.1 Definition

Box Types: `'url '`, `'urn '`, `'dref'`
Container: Data Information Box (`'dinf'`)
Mandatory: Yes
Quantity: Exactly one

The data reference object contains a table of data references (normally URLs) that declare the location(s) of the media data used within the presentation. The data reference index in the sample description ties entries in this table to the samples in the track. A track may be split over several sources in this way.

If the flag is set indicating that the data is in the same file as this box, then no string (not even an empty one) shall be supplied in the entry field.

The DataEntryBox within the DataReferenceBox shall be either a DataEntryUrnBox or a DataEntryUrlBox.

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

When a file that has data entries with the flag set indicating that the media data is in the same file, is split into segments for transport, the value of this flag does not change, as the file is (logically) reassembled after the transport operation.

8.7.2.2 Syntax

```
aligned(8) class DataEntryUrlBox (bit(24) flags)
   extends FullBox('url ', version = 0, flags) {
   string   location;
}


aligned(8) class DataEntryUrnBox (bit(24) flags)
   extends FullBox('urn ', version = 0, flags) {
   string   name;
   string   location;
}


aligned(8) class DataReferenceBox
   extends FullBox('dref', version = 0, 0) {
   unsigned int(32)  entry_count;
   for (i=1; i <= entry_count; i++) {
      DataEntryBox(entry_version, entry_flags) data_entry;
   }
}
```

8.7.2.3 Semantics

`version` is an integer that specifies the version of this box
`entry_count` is an integer that counts the actual entries
`entry_version` is an integer that specifies the version of the entry format
`entry_flags` is a 24-bit integer with flags; one flag is defined (x000001) which means that the media data is in the same file as the Movie Box containing this data reference.
`data_entry` is a URL or URN entry. Name is a URN, and is required in a URN entry. Location is a URL, and is required in a URL entry and optional in a URN entry, where it gives a location to find the resource with the given name. Each is a null-terminated string using UTF-8 characters. If the self-contained flag is set, the URL form is used and no string is present; the box terminates with the entry-flags field. The URL type should be of a service that delivers a file (e.g. URLs of type file, http, ftp etc.), and which services ideally also permit random access. Relative URLs are permissible and are relative to the file containing the Movie Box that contains this data reference.

8.7.3 Sample Size Boxes

8.7.3.1 Definition

Box Type: `'stsz'`,`'stz2'`
Container: Sample Table Box (`'stbl'`)
Mandatory: Yes
Quantity: Exactly one variant must be present

This box contains the sample count and a table giving the size in bytes of each sample. This allows the media data itself to be unframed. The total number of samples in the media is always indicated in the sample count.

There are two variants of the sample size box. The first variant has a fixed size 32-bit field for representing the sample sizes; it permits defining a constant size for all samples in a track. The second variant permits smaller size fields, to save space when the sizes are varying but small. One of these boxes must be present; the first version is preferred for maximum compatibility.

NOTE A sample size of zero is not prohibited in general, but it must be valid and defined for the coding system, as defined by the sample entry, that the sample belongs to.

8.7.3.2 Sample Size Box

8.7.3.2.1 Syntax

```
aligned(8) class SampleSizeBox extends FullBox('stsz', version = 0, 0) {
   unsigned int(32)  sample_size;
   unsigned int(32)  sample_count;
   if (sample_size==0) {
      for (i=1; i <= sample_count; i++) {
      unsigned int(32)  entry_size;
      }
   }
}
```

8.7.3.2.2 Semantics

`version` is an integer that specifies the version of this box
`sample_size` is integer specifying the default sample size. If all the samples are the same size, this field contains that size value. If this field is set to 0, then the samples have different sizes, and those sizes are stored in the sample size table. If this field is not 0, it specifies the constant sample size, and no array follows.
`sample_count` is an integer that gives the number of samples in the track; if sample-size is 0, then it is also the number of entries in the following table.
`entry_size` is an integer specifying the size of a sample, indexed by its number.

8.7.3.3 Compact Sample Size Box

8.7.3.3.1 Syntax

```
aligned(8) class CompactSampleSizeBox extends FullBox('stz2', version = 0, 0) {
   unsigned int(24)  reserved = 0;
   unisgned int(8)   field_size;
   unsigned int(32)  sample_count;
   for (i=1; i <= sample_count; i++) {
      unsigned int(field_size)   entry_size;
   }
}
```

8.7.3.3.2 Semantics

`version` is an integer that specifies the version of this box
`field_size` is an integer specifying the size in bits of the entries in the following table; it shall take the value 4, 8 or 16. If the value 4 is used, then each byte contains two values: entry[i]<<4 + entry[i+1]; if the sizes do not fill an integral number of bytes, the last byte is padded with zeros.
`sample_count` is an integer that gives the number of entries in the following table
`entry_size` is an integer specifying the size of a sample, indexed by its number.

8.7.4 Sample To Chunk Box

8.7.4.1 Definition

Box Type: `'stsc'`
Container: Sample Table Box (`'stbl'`)
Mandatory: Yes
Quantity: Exactly one

Samples within the media data are grouped into chunks. Chunks can be of different sizes, and the samples within a chunk can have different sizes. This table can be used to find the chunk that contains a sample, its position, and the associated sample description.

The table is compactly coded. Each entry gives the index of the first chunk of a run of chunks with the same characteristics. By subtracting one entry here from the previous one, you can compute how many chunks are in this run. You can convert this to a sample count by multiplying by the appropriate samples-per-chunk.

8.7.4.2 Syntax

```
aligned(8) class SampleToChunkBox
   extends FullBox('stsc', version = 0, 0) {
   unsigned int(32)  entry_count;
   for (i=1; i <= entry_count; i++) {
      unsigned int(32)  first_chunk;
      unsigned int(32)  samples_per_chunk;
      unsigned int(32)  sample_description_index;
   }
}
```

8.7.4.3 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
`first_chunk` is an integer that gives the index of the first chunk in this run of chunks that share the same samples-per-chunk and sample-description-index; the index of the first chunk in a track has the value 1 (the `first_chunk` field in the first record of this box has the value 1, identifying that the first sample maps to the first chunk).
`samples_per_chunk` is an integer that gives the number of samples in each of these chunks
`sample_description_index` is an integer that gives the index of the sample entry that describes the samples in this chunk. The index ranges from 1 to the number of sample entries in the Sample Description Box

8.7.5 Chunk Offset Box

8.7.5.1 Definition

Box Type: `'stco'`, `'co64'`
Container: Sample Table Box (`'stbl'`)
Mandatory: Yes
Quantity: Exactly one variant must be present

The chunk offset table gives the index of each chunk into the containing file. There are two variants, permitting the use of 32-bit or 64-bit offsets. The latter is useful when managing very large presentations. At most one of these variants will occur in any single instance of a sample table.

Offsets are file offsets, not the offset into any box within the file (e.g. Media Data Box). This permits referring to media data in files without any box structure. It does also mean that care must be taken when constructing a self-contained ISO file with its metadata (Movie Box) at the front, as the size of the Movie Box will affect the chunk offsets to the media data.

8.7.5.2 Syntax

```
aligned(8) class ChunkOffsetBox
   extends FullBox('stco', version = 0, 0) {
   unsigned int(32)  entry_count;
   for (i=1; i <= entry_count; i++) {
      unsigned int(32)  chunk_offset;
   }
}


aligned(8) class ChunkLargeOffsetBox
   extends FullBox('co64', version = 0, 0) {
   unsigned int(32)  entry_count;
   for (i=1; i <= entry_count; i++) {
      unsigned int(64)  chunk_offset;
   }
}
```

8.7.5.3 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
`chunk_offset` is a 32 or 64 bit integer that gives the offset of the start of a chunk into its containing media file.

8.7.6 Padding Bits Box

8.7.6.1 Definition

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

In some streams the media samples do not occupy all bits of the bytes given by the sample size, and are padded at the end to a byte boundary. In some cases, it is necessary to record externally the number of padding bits used. This table supplies that information.

8.7.6.2 Syntax

```
aligned(8) class PaddingBitsBox extends FullBox('padb', version = 0, 0) {
   unsigned int(32) sample_count;
   int i;
   for (i=0; i < ((sample_count + 1)/2); i++) {
      bit(1)   reserved = 0;
      bit(3)   pad1;
      bit(1)   reserved = 0;
      bit(3)   pad2;
   }
}
```

8.7.6.3 Semantics

`sample_count` – counts the number of samples in the track; it should match the count in other tables
`pad1` – a value from 0 to 7, indicating the number of bits at the end of sample (i*2)+1.
`pad2` – a value from 0 to 7, indicating the number of bits at the end of sample (i*2)+2

8.7.7 Sub-Sample Information Box

8.7.7.1 Definition

Box Type: 'subs'
Container: Sample Table Box ('stbl') or Track Fragment Box ('traf')
Mandatory: No
Quantity: Zero or one

This box, named the *Sub-Sample Information box*, is designed to contain sub-sample information.

A sub-sample is a contiguous range of bytes of a sample. The specific definition of a sub-sample shall be supplied for a given coding system (e.g. for ISO/IEC 14496-10, Advanced Video Coding). In the absence of such a specific definition, this box shall not be applied to samples using that coding system.

If subsample_count is 0 for any entry, then those samples have no subsample information and no array follows. The table is sparsely coded; the table identifies which samples have sub-sample structure by recording the difference in sample-number between each entry. The first entry in the table records the sample number of the first sample having sub-sample information.

NOTE It is possible to combine subsample_priority and discardable such that when subsample_priority is smaller than a certain value, discardable is set to 1. However, since different systems may use different scales of priority values, to separate them is safe to have a clean solution for discardable sub-samples.

8.7.7.2 Syntax

```
aligned(8) class SubSampleInformationBox
   extends FullBox('subs', version, 0) {
   unsigned int(32) entry_count;
   int i,j;
   for (i=0; i < entry_count; i++) {
      unsigned int(32) sample_delta;
      unsigned int(16) subsample_count;
      if (subsample_count > 0) {
         for (j=0; j < subsample_count; j++) {
            if(version == 1)
            {
               unsigned int(32) subsample_size;
            }
            else
            {
               unsigned int(16) subsample_size;
            }
            unsigned int(8) subsample_priority;
            unsigned int(8) discardable;
            unsigned int(32) reserved = 0;
         }
      }
   }
}
```

8.7.7.3 Semantics

`version` is an integer that specifies the version of this box (0 or 1 in this specification)
`entry_count` is an integer that gives the number of entries in the following table.
`sample_delta` is an integer that specifies the sample number of the sample having sub-sample structure. It is coded as the difference between the desired sample number, and the sample number indicated in the previous entry. If the current entry is the first entry, the value indicates the sample number of the first sample having sub-sample information, that is, the value is the difference between the sample number and zero (0).
`subsample_count` is an integer that specifies the number of sub-sample for the current sample. If there is no sub-sample structure, then this field takes the value 0.
`subsample_size` is an integer that specifies the size, in bytes, of the current sub-sample.
`subsample_priority` is an integer specifying the degradation priority for each sub-sample. Higher values of `subsample_priority`, indicate sub-samples which are important to, and have a greater impact on, the decoded quality.
`discardable` equal to 0 means that the sub-sample is required to decode the current sample, while equal to 1 means the sub-sample is not required to decode the current sample but may be used for enhancements, e.g., the sub-sample consists of supplemental enhancement information (SEI) messages.

8.7.8 Sample Auxiliary Information Sizes Box

8.7.8.1 Definition

Box Type: `'saiz'`
Container: Sample Table Box (`'stbl'`) or Track Fragment Box (`'traf'`)
Mandatory: No
Quantity: Zero or More

Per-sample sample auxiliary information may be stored anywhere in the same file as the sample data itself; for self-contained media files, this is typically in a MediaData box or a box from a derived specification. It is stored either (a) in multiple chunks, with the number of samples per chunk, as well as the number of chunks, matching the chunking of the primary sample data or (b) in a single chunk for all the samples in a movie sample table (or a movie fragment). The Sample Auxiliary Information for all samples contained within a single chunk (or track run) is stored contiguously (similarly to sample data).

Sample Auxiliary Information, when present, is always stored in the same file as the samples to which it relates as they share the same data reference (`'dref'`) structure. However, this data may be located anywhere within this file, using auxiliary information offsets (`'saio'`) to indicate the location of the data.

Whether sample auxiliary information is permitted or required may be specified by the brands or the coding format in use. The format of the sample auxiliary information is determined by `aux_info_type`. If `aux_info_type` and `aux_info_type_parameter` are omitted then the implied value of `aux_info_type` is either (a) in the case of transformed content, such as protected content, the `scheme_type` included in the Protection Scheme Information box or otherwise (b) the sample entry type. The default value of the `aux_info_type_parameter` is 0. Some values of `aux_info_type` may be restricted to be used only with particular track types. A track may have multiple streams of sample auxiliary information of different types. The types are registered at the registration authority.

While `aux_info_type` determines the format of the auxiliary information, several streams of auxiliary information having the same format may be used when their value of `aux_info_type_parameter` differs. The semantics of `aux_info_type_parameter` for a particular `aux_info_type` value must be specified along with specifying the semantics of the particular `aux_info_type` value and the implied auxiliary information format.

This box provides the size of the auxiliary information for each sample. For each instance of this box, there must be a matching `SampleAuxiliaryInformationOffsetsBox` with the same values of `aux_info_type` and `aux_info_type_parameter`, providing the offset information for this auxiliary information.

NOTE For discussions on the use of sample auxiliary information versus other mechanisms, see Annex C.8.

8.7.8.2 Syntax

```
aligned(8) class SampleAuxiliaryInformationSizesBox
   extends FullBox('saiz', version = 0, flags)
{
   if (flags & 1) {
      unsigned int(32) aux_info_type;
      unsigned int(32) aux_info_type_parameter;
   }
   unsigned int(8) default_sample_info_size;
   unsigned int(32) sample_count;
   if (default_sample_info_size == 0) {
      unsigned int(8) sample_info_size[ sample_count ];
   }
}
```

8.7.8.3 Semantics

`aux_info_type` is an integer that identifies the type of the sample auxiliary information. At most one occurrence of this box with the same values for `aux_info_type` and `aux_info_type_parameter` shall exist in the containing box.

`aux_info_type_parameter` identifies the "stream" of auxiliary information having the same value of `aux_info_type` and associated to the same track. The semantics of `aux_info_type_parameter` are determined by the value of `aux_info_type`.

`default_sample_info_size` is an integer specifying the sample auxiliary information size for the case where all the indicated samples have the same sample auxiliary information size. If the size varies then this field shall be zero.

`sample_count` is an integer that gives the number of samples for which a size is defined. For a Sample Auxiliary Information Sizes box appearing in the Sample Table Box this must be the same as, or less than, the `sample_count` within the Sample Size Box or Compact Sample Size Box. For a Sample Auxiliary Information Sizes box appearing in a Track Fragment box this must be the same as, or less than, the sum of the `sample_count` entries within the Track Fragment Run boxes of the Track Fragment. If this is less than the number of samples, then auxiliary information is supplied for the initial samples, and the remaining samples have no associated auxiliary information.

`sample_info_size` gives the size of the sample auxiliary information in bytes. This may be zero to indicate samples with no associated auxiliary information.

8.7.9 Sample Auxiliary Information Offsets Box

8.7.9.1 Definition

Box Type: `'saio'`
Container: Sample Table Box (`'stbl'`) or Track Fragment Box (`'traf'`)
Mandatory: No
Quantity: Zero or More

For an introduction to sample auxiliary information, see the definition of the Sample Auxiliary Information Size Box.

This box provides the position information for the sample auxiliary information, in a way similar to the chunk offsets for sample data.

8.7.9.2 Syntax

```
aligned(8) class SampleAuxiliaryInformationOffsetsBox
   extends FullBox('saio', version, flags)
{
   if (flags & 1) {
      unsigned int(32) aux_info_type;
      unsigned int(32) aux_info_type_parameter;
   }
   unsigned int(32) entry_count;
   if ( version == 0 ) {
      unsigned int(32) offset[ entry_count ];
   }
   else {
      unsigned int(64) offset[ entry_count ];
   }
}
```

8.7.9.3 Semantics

`aux_info_type` and `aux_info_type_parameter` are defined as in the `SampleAuxiliaryInformationSizesBox`

`entry_count` gives the number of entries in the following table. For a Sample Auxiliary Information Offsets box appearing in a Sample Table Box this must be equal to one or to the value of the `entry_count` field in the Chunk Offset Box or Chunk Large Offset Box. For a Sample Auxiliary Information Offsets Box appearing in a Track Fragment box, this must be equal to one or to the number of Track Fragment Run boxes in the Track Fragment Box.

`offset` gives the position in the file of the Sample Auxiliary Information for each Chunk or Track Fragment Run. If `entry_count` is one, then the Sample Auxiliary Information for all Chunks or Runs is contiguous in the file in chunk or run order. When in the Sample Table Box, the offsets are absolute. In a track fragment box, this value is relative to the base offset established by the track fragment header box (`'tfhd'`) in the same track fragment (see 8.8.14).

8.8 Movie Fragments

8.8.1 Movie Extends Box

8.8.1.1 Definition

Box Type: `'mvex'`
Container: Movie Box (`'moov'`)
Mandatory: No
Quantity: Zero or one

This box warns readers that there might be Movie Fragment Boxes in this file. To know of all samples in the tracks, these Movie Fragment Boxes must be found and scanned in order, and their information logically added to that found in the Movie Box.

There is a narrative introduction to Movie Fragments in Annex A.

8.8.1.2 Syntax

```
aligned(8) class MovieExtendsBox extends Box('mvex'){
}
```

8.8.2 Movie Extends Header Box

8.8.2.1 Definition

Box Type: 'mehd'
Container: Movie Extends Box('mvex')
Mandatory: No
Quantity: Zero or one

The Movie Extends Header is optional, and provides the overall duration, including fragments, of a fragmented movie. If this box is not present, the overall duration must be computed by examining each fragment.

8.8.2.2 Syntax

```
aligned(8) class MovieExtendsHeaderBox extends FullBox('mehd', version, 0) {
   if (version==1) {
      unsigned int(64)  fragment_duration;
   } else { // version==0
      unsigned int(32)  fragment_duration;
   }
}
```

8.8.2.3 Semantics

`fragment_duration` is an integer that declares length of the presentation of the whole movie including fragments (in the timescale indicated in the Movie Header Box). The value of this field corresponds to the duration of the longest track, including movie fragments. If an MP4 file is created in real-time, such as used in live streaming, it is not likely that the `fragment_duration` is known in advance and this box may be omitted.

8.8.3 Track Extends Box

8.8.3.1 Definition

Box Type: 'trex'
Container: Movie Extends Box ('mvex')
Mandatory: Yes
Quantity: Exactly one for each track in the Movie Box

This sets up default values used by the movie fragments. By setting defaults in this way, space and complexity can be saved in each Track Fragment Box.

The sample flags field in sample fragments (`default_sample_flags` here and in a Track Fragment Header Box, and `sample_flags` and `first_sample_flags` in a Track Fragment Run Box) is coded as a 32-bit value. It has the following structure:

```
bit(4)   reserved=0;
unsigned int(2) is_leading;
unsigned int(2) sample_depends_on;
unsigned int(2) sample_is_depended_on;
unsigned int(2) sample_has_redundancy;
bit(3)   sample_padding_value;
bit(1)   sample_is_non_sync_sample;
unsigned int(16)  sample_degradation_priority;
```

The `is_leading`, `sample_depends_on`, `sample_is_depended_on` and `sample_has_redundancy` values are defined as documented in the Independent and Disposable Samples Box.

The flag `sample_is_non_sync_sample` provides the same information as the sync sample table [8.6.2]. When this value is set 0 for a sample, it is the same as if the sample were not in a movie fragment and marked with an entry in the sync sample table (or, if all samples are sync samples, the sync sample table were absent).

The `sample_padding_value` is defined as for the padding bits table. The `sample_degradation_priority` is defined as for the degradation priority table.

8.8.3.2 Syntax

```
aligned(8) class TrackExtendsBox extends FullBox('trex', 0, 0){
   unsigned int(32)  track_ID;
   unsigned int(32)  default_sample_description_index;
   unsigned int(32)  default_sample_duration;
   unsigned int(32)  default_sample_size;
   unsigned int(32)  default_sample_flags
}
```

8.8.3.3 Semantics

`track_id` identifies the track; this shall be the track ID of a track in the Movie Box
`default_` these fields set up defaults used in the track fragments.

8.8.4 Movie Fragment Box

8.8.4.1 Definition

Box Type: `'moof'`
Container: File
Mandatory: No
Quantity: Zero or more

The movie fragments extend the presentation in time. They provide the information that would previously have been in the Movie Box. The actual samples are in Media Data Boxes, as usual, if they are in the same file. The data reference index is in the sample description, so it is possible to build incremental presentations where the media data is in files other than the file containing the Movie Box.

The Movie Fragment Box is a top-level box, (i.e. a peer to the Movie Box and Media Data boxes). It contains a Movie Fragment Header Box, and then one or more Track Fragment Boxes.

> NOTE There is no requirement that any particular movie fragment extend all tracks present in the movie header, and there is no restriction on the location of the media data referred to by the movie fragments. However, derived specifications may make such restrictions.

8.8.4.2 Syntax

```
aligned(8) class MovieFragmentBox extends Box('moof'){
}
```

8.8.5 Movie Fragment Header Box

8.8.5.1 Definition

Box Type: `'mfhd'`
Container: Movie Fragment Box (`'moof'`)
Mandatory: Yes
Quantity: Exactly one

The movie fragment header contains a sequence number, as a safety check. The sequence number usually starts at 1 and must increase for each movie fragment in the file, in the order in which they occur. This allows readers to verify integrity of the sequence; it is an error to construct a file where the fragments are out of sequence.

NOTE There is no requirement that the sequence numbers be consecutive, only that the value in a given movie fragment be greater than in any preceding movie fragment.

8.8.5.2 Syntax

```
aligned(8) class MovieFragmentHeaderBox
         extends FullBox('mfhd', 0, 0){
   unsigned int(32)  sequence_number;
}
```

8.8.5.3 Semantics

`sequence_number` the ordinal number of this fragment, in increasing order

8.8.6 Track Fragment Box

8.8.6.1 Definition

Box Type: `'traf'`
Container: Movie Fragment Box (`'moof'`)
Mandatory: No
Quantity: Zero or more

Within the movie fragment there is a set of track fragments, zero or more per track. The track fragments in turn contain zero or more track runs, each of which document a contiguous run of samples for that track. Within these structures, many fields are optional and can be defaulted.

It is possible to add 'empty time' to a track using these structures, as well as adding samples. Empty inserts can be used in audio tracks doing silence suppression, for example.

8.8.6.2 Syntax

```
aligned(8) class TrackFragmentBox extends Box('traf'){
}
```

8.8.7 Track Fragment Header Box

8.8.7.1 Definition

Box Type: `'tfhd'`
Container: Track Fragment Box (`'traf'`)
Mandatory: Yes
Quantity: Exactly one

Each movie fragment can add zero or more fragments to each track; and a track fragment can add zero or more contiguous runs of samples. The track fragment header sets up information and defaults used for those runs of samples.

The following flags are defined in the `tf_flags`:

`0x000001` base-data-offset-present: indicates the presence of the base-data-offset field. This provides an explicit anchor for the data offsets in each track run (see below). If not provided, the base-data-offset for the first track in the movie fragment is the position of the first byte of the enclosing Movie Fragment Box, and for second and subsequent track fragments, the default is the end of the data defined by the preceding fragment. Fragments 'inheriting' their offset in this way must all use the same data-reference (i.e., the data for these tracks must be in the same file).

`0x000002` sample-description-index-present: indicates the presence of this field, which over-rides, in this fragment, the default set up in the Track Extends Box.

`0x000008` default-sample-duration-present

`0x000010` default-sample-size-present

`0x000020` default-sample-flags-present

`0x010000` duration-is-empty: this indicates that the duration provided in either default-sample-duration, or by the default-duration in the Track Extends Box, is empty, i.e. that there are no samples for this time interval. It is an error to make a presentation that has both edit lists in the Movie Box, and empty-duration fragments.

`0x020000` default-base-is-moof: if base-data-offset-present is zero, this indicates that the base-data-offset for this track fragment is the position of the first byte of the enclosing Movie Fragment Box. Support for the default-base-is-moof flag is required under the ‘iso5’ brand, and it shall not be used in brands or compatible brands earlier than iso5.

NOTE The use of the default-base-is-moof flag breaks the compatibility to earlier brands of the file format, because it sets the anchor point for offset calculation differently than earlier. Therefore, the default-base-is-moof flag cannot be set when earlier brands are included in the File Type box.

8.8.7.2 Syntax

```
aligned(8) class TrackFragmentHeaderBox
         extends FullBox('tfhd', 0, tf_flags){
   unsigned int(32)  track_ID;
   // all the following are optional fields
   unsigned int(64)  base_data_offset;
   unsigned int(32)  sample_description_index;
   unsigned int(32)  default_sample_duration;
   unsigned int(32)  default_sample_size;
   unsigned int(32)  default_sample_flags
}
```

8.8.7.3 Semantics

`base_data_offset` the base offset to use when calculating data offsets

8.8.8 Track Fragment Run Box

8.8.8.1 Definition

Box Type: `'trun'`
Container: Track Fragment Box (`'traf'`)
Mandatory: No
Quantity: Zero or more

Within the Track Fragment Box, there are zero or more Track Run Boxes. If the duration-is-empty flag is set in the `tf_flags`, there are no track runs. A track run documents a contiguous set of samples for a track.

The number of optional fields is determined from the number of bits set in the lower byte of the flags, and the size of a record from the bits set in the second byte of the flags. This procedure shall be followed, to allow for new fields to be defined.

If the data-offset is not present, then the data for this run starts immediately after the data of the previous run, or at the base-data-offset defined by the track fragment header if this is the first run in a track fragment, If the data-offset is present, it is relative to the base-data-offset established in the track fragment header.

The following flags are defined:

0x000001 data-offset-present.
0x000004 first-sample-flags-present; this over-rides the default flags for the first sample only. This makes it possible to record a group of frames where the first is a key and the rest are difference frames, without supplying explicit flags for every sample. If this flag and field are used, sample-flags shall not be present.
0x000100 sample-duration-present: indicates that each sample has its own duration, otherwise the default is used.
0x000200 sample-size-present: each sample has its own size, otherwise the default is used.
0x000400 sample-flags-present; each sample has its own flags, otherwise the default is used.
0x000800 sample-composition-time-offsets-present; each sample has a composition time offset (e.g. as used for I/P/B video in MPEG).

The composition offset values in the composition time-to-sample box and in the track run box may be signed or unsigned. The recommendations given in the composition time-to-sample box concerning the use of signed composition offsets also apply here.

8.8.8.2 Syntax

```
aligned(8) class TrackRunBox
         extends FullBox('trun', version, tr_flags) {
   unsigned int(32)  sample_count;
   // the following are optional fields
   signed int(32) data_offset;
   unsigned int(32)  first_sample_flags;
   // all fields in the following array are optional
   {
      unsigned int(32)  sample_duration;
      unsigned int(32)  sample_size;
      unsigned int(32)  sample_flags
      if (version == 0)
         { unsigned int(32)    sample_composition_time_offset; }
      else
         { signed int(32)      sample_composition_time_offset; }
   }[ sample_count ]
}
```

8.8.8.3 Semantics

`sample_count` the number of samples being added in this run; also the number of rows in the following table (the rows can be empty)
`data_offset` is added to the implicit or explicit `data_offset` established in the track fragment header.
`first_sample_flags` provides a set of flags for the first sample only of this run.

8.8.9 Movie Fragment Random Access Box

8.8.9.1 Definition

Box Type: `'mfra'`
Container: File
Mandatory: No
Quantity: Zero or one

The Movie Fragment Random Access Box (‘mfra’) provides a table which may assist readers in finding sync samples in a file using movie fragments. It contains a track fragment random access box for each track for which information is provided (which may not be all tracks). It is usually placed at or near the end of the file; the last box within the Movie Fragment Random Access Box provides a copy of the length field from the Movie Fragment Random Access Box. Readers may attempt to find this box by examining the last 32 bits of the file, or scanning backwards from the end of the file for a Movie Fragment Random Access Offset Box and using the size information in it, to see if that locates the beginning of a Movie Fragment Random Access Box.

This box provides only a hint as to where sync samples are; the movie fragments themselves are definitive. It is recommended that readers take care in both locating and using this box as modifications to the file after it was created may render either the pointers, or the declaration of sync samples, incorrect.

8.8.9.2 Syntax

```
aligned(8) class MovieFragmentRandomAccessBox
 extends Box(‘mfra’)
{
}
```

8.8.10 Track Fragment Random Access Box

8.8.10.1 Definition

Box Type: ‘tfra’
Container: Movie Fragment Random Access Box (‘mfra’)
Mandatory: No
Quantity: Zero or one per track

Each entry contains the location and the presentation time of the sync sample. Note that not every sync sample in the track needs to be listed in the table.

The absence of this box does not mean that all the samples are sync samples. Random access information in the ‘trun’, ‘traf’ and ‘trex’ shall be set appropriately regardless of the presence of this box.

8.8.10.2 Syntax

```
aligned(8) class TrackFragmentRandomAccessBox
 extends FullBox(‘tfra’, version, 0) {
   unsigned int(32)  track_ID;
   const unsigned int(26)  reserved = 0;
   unsigned int(2)   length_size_of_traf_num;
   unsigned int(2)   length_size_of_trun_num;
   unsigned int(2)   length_size_of_sample_num;
   unsigned int(32)  number_of_entry;
   for(i=1; i <= number_of_entry; i++){
      if(version==1){
         unsigned int(64)  time;
         unsigned int(64)  moof_offset;
      }else{
         unsigned int(32)  time;
         unsigned int(32)  moof_offset;
      }
      unsigned int((length_size_of_traf_num+1) * 8)   traf_number;
      unsigned int((length_size_of_trun_num+1) * 8)   trun_number;
      unsigned int((length_size_of_sample_num+1) * 8) sample_number;
   }
}
```

8.8.10.3 Semantics

`track_ID` is an integer identifying the `track_ID`.
`length_size_of_traf_num` indicates the length in byte of the `traf_number` field minus one.
`length_size_of_trun_num` indicates the length in byte of the `trun_number` field minus one.
`length_size_of_sample_num` indicates the length in byte of the `sample_number` field minus one.
`number_of_entry` is an integer that gives the number of the entries for this track. If this value is zero, it indicates that every sample is a sync sample and no table entry follows.
`time` is 32 or 64 bits integer that indicates the presentation time of the sync sample in units defined in the `'mdhd'` of the associated track.
`moof_offset` is 32 or 64 bits integer that gives the offset of the `'moof'` used in this entry. Offset is the byte-offset between the beginning of the file and the beginning of the `'moof'`.
`traf_number` indicates the `'traf'` number that contains the sync sample. The number ranges from 1 (the first `'traf'` is numbered 1) in each `'moof'`.
`trun_number` indicates the `'trun'` number that contains the sync sample. The number ranges from 1 in each `'traf'`.
`sample_number` indicates the sample number of the sync sample. The number ranges from 1 in each `'trun'`.

8.8.11 Movie Fragment Random Access Offset Box

8.8.11.1 Definition

Box Type: `'mfro'`
Container: Movie Fragment Random Access Box (`'mfra'`)
Mandatory: Yes
Quantity: Exactly one

The Movie Fragment Random Access Offset Box provides a copy of the length field from the enclosing Movie Fragment Random Access Box. It is placed last within that box, so that the size field is also last in the enclosing Movie Fragment Random Access Box. When the Movie Fragment Random Access Box is also last in the file this permits its easy location. The size field here must be correct. However, neither the presence of the Movie Fragment Random Access Box, nor its placement last in the file, are assured.

8.8.11.2 Syntax

```
aligned(8) class MovieFragmentRandomAccessOffsetBox
 extends FullBox('mfro', version, 0) {
   unsigned int(32)  size;
}
```

8.8.11.3 Semantics

`size` is an integer gives the number of bytes of the enclosing `'mfra'` box. This field is placed at the last of the enclosing box to assist readers scanning from the end of the file in finding the `'mfra'` box.

8.8.12 Track fragment decode time

8.8.12.1 Definition

Box Type: `` `tfdt' ``
Container: Track Fragment box (`'traf'`)
Mandatory: No
Quantity: Zero or one

The Track Fragment Base Media Decode Time Box provides the absolute decode time, measured on the media timeline, of the first sample in decode order in the track fragment. This can be useful, for example, when performing random access in a file; it is not necessary to sum the sample durations of all preceding samples in previous fragments to find this value (where the sample durations are the deltas in the Decoding Time to Sample Box and the sample_durations in the preceding track runs).

The Track Fragment Base Media Decode Time Box, if present, shall be positioned after the Track Fragment Header Box and before the first Track Fragment Run box.

NOTE The decode timeline is a media timeline, established before any explicit or implied mapping of media time to presentation time, for example by an edit list or similar structure.

8.8.12.2 Syntax

```
aligned(8) class TrackFragmentBaseMediaDecodeTimeBox
   extends FullBox('tfdt', version, 0) {
   if (version==1) {
      unsigned int(64) baseMediaDecodeTime;
   } else { // version==0
      unsigned int(32) baseMediaDecodeTime;
   }
}
```

8.8.12.3 Semantics

`version` is an integer that specifies the version of this box (0 or 1 in this specification).
`baseMediaDecodeTime` is an integer equal to the sum of the decode durations of all earlier samples in the media, expressed in the media's timescale. It does not include the samples added in the enclosing track fragment.

8.8.13 Level Assignment Box

8.8.13.1 Definition

Box Type: `leva'
Container: Movie Extends Box (`mvex')
Mandatory: No
Quantity: Zero or one

Levels specify subsets of the file. Samples mapped to level n may depend on any samples of levels m, where m <= n, and shall not depend on any samples of levels p, where p > n. For example, levels can be specified according to temporal level (e.g., temporal_id of SVC or MVC).

Levels cannot be specified for the initial movie. When the Level Assignment box is present, it applies to all movie fragments subsequent to the initial movie.

For the context of the Level Assignment box, a fraction is defined to consist of one or more Movie Fragment boxes and the associated Media Data boxes, possibly including only an initial part of the last Media Data Box. Within a fraction, data for each level shall appear contiguously. Data for levels within a fraction shall appear in increasing order of level value. All data in a fraction shall be assigned to levels.

NOTE In the context of DASH (ISO/IEC 23009-1), each subsegment indexed within a Subsegment Index box is a fraction.

The Level Assignment box provides a mapping from features, such as scalability layers, to levels. A feature can be specified through a track, a sub-track within a track, or a sample grouping of a track.

When padding_flag is equal to 1 this indicates that a conforming fraction can be formed by concatenating any positive integer number of levels within a fraction and padding the last Media Data box by zero bytes up to the full size that is indicated in the header of the last Media Data box. For example, `padding_flag` can be set equal to 1 when the following conditions are true:

- Each fraction contains two or more AVC, SVC, or MVC [ISO/IEC 14496-15] tracks of the same video bitstream.
- The samples for each track of a fraction are contiguous and in decoding order in a Media Data box.
- The samples of the first AVC, SVC, or MVC level contain extractor NAL units for including the video coding NAL units from the other levels of the same fraction.

8.8.13.2 Syntax

```
aligned(8) class LevelAssignmentBox extends FullBox('leva', 0, 0)
{
   unsigned int(8)   level_count;
   for (j=1; j <= level_count; j++) {
      unsigned int(32)  track_id;
      unsigned int(1)   padding_flag;
      unsigned int(7)   assignment_type;
      if (assignment_type == 0) {
         unsigned int(32)  grouping_type;
      }
      else if (assignment_type == 1) {
         unsigned int(32)  grouping_type;
         unsigned int(32)  grouping_type_parameter;
      }
      else if (assignment_type == 2) {} // no further syntax elements needed
      else if (assignment_type == 3) {} // no further syntax elements needed
      else if (assignment_type == 4) {
         unsigned int(32) sub_track_id;
      }
      // other assignment_type values are reserved
   }
}
```

8.8.13.3 Semantics

`level_count` specifies the number of levels each fraction is grouped into. `level_count` shall be greater than or equal to 2.

`track_id` for loop entry j specifies the track identifier of the track assigned to level j.

`padding_flag` equal to 1 indicates that a conforming fraction can be formed by concatenating any positive integer number of levels within a fraction and padding the last Media Data box by zero bytes up to the full size that is indicated in the header of the last Media Data box. The semantics of `padding_flag` equal to 0 are that this is not assured.

`assignment_type` indicates the mechanism used to specify the assignment to a level. `assignment_type` values greater than 4 are reserved, while the semantics for the other values are specified as follows. The sequence of assignment_types is restricted to be a set of zero or more of type 2 or 3, followed by zero or more of exactly one type.

- 0: sample groups are used to specify levels, i.e., samples mapped to different sample group description indexes of a particular sample grouping lie in different levels within the identified track; other tracks are not affected and must have all their data in precisely one level;
- 1: as for assignment_type 0 except assignment is by a parameterized sample group;
- 2, 3: level assignment is by track (see the Subsegment Index Box for the difference in processing of these levels)
- 4: the respective level contains the samples for a sub-track. The sub-tracks are specified through the Sub Track box; other tracks are not affected and must have all their data in precisely one level;

`grouping_type` and `grouping_type_parameter`, if present, specify the sample grouping used to map sample group description entries in the Sample Group Description box to levels. Level n contains the samples that are mapped to the sample group description entry having index n in the Sample Group Description box having the same values of `grouping_type` and `grouping_type_parameter`, if present, as those provided in this box.

`sub_track_id` specifies that the sub-track identified by `sub_track_id` within loop entry j is mapped to level j.

8.8.14 Sample Auxiliary Information in Movie Fragments

When sample auxiliary information (8.7.8 and 8.7.9) is present in the Movie Fragment box, the offsets in the Sample Auxiliary Information Offsets Box are treated the same as the `data_offset` in the Track Fragment Run box, that is, they are relative to any base data offset established for that track fragment. If movie fragment relative addressing is used (no base data offset is provided in the track fragment header) and auxiliary information is present, then the `default_base_is_moof` flag must also be set in the flags of that track fragment header.

If only one offset is provided, then the Sample Auxiliary Information for all the track runs in the fragment is stored contiguously, otherwise exactly one offset must be provided for each track run.

If the field `default_sample_info_size` is non-zero in one of these boxes, then the size of the auxiliary information is constant for the identified samples.

In addition, if:

- this box is present in the movie box,
- and `default_sample_info_size` is non-zero in the box in the movie box,
- and the sample auxiliary information sizes box is absent in a movie fragment,

then the auxiliary information has this same constant size for every sample in the movie fragment also; it is then not necessary to repeat the box in the movie fragment.

8.9 Sample Group Structures

8.9.1 Introduction

This clause specifies a generic mechanism for representing a partition of the samples in a track. A *sample grouping* is an assignment of each sample in a track to be a member of one *sample group,* based on a grouping criterion. A sample group in a sample grouping is not limited to being contiguous samples and may contain non-adjacent samples. As there may be more than one sample grouping for the samples in a track, each sample grouping has a type field to indicate the type of grouping. For example, a file might contain two sample groupings for the same track: one based on an assignment of sample to layers and another to sub-sequences.

Sample groupings are represented by two linked data structures: (1) a `SampleToGroup` box represents the assignment of samples to sample groups; (2) a `SampleGroupDescription` box contains a *sample group entry* for each sample group describing the properties of the group. There may be multiple instances of the `SampleToGroup` and `SampleGroupDescription` boxes based on different grouping criteria. These are distinguished by a type field used to indicate the type of grouping.

A grouping of a particular grouping type may use a parameter in the sample to group mapping; if so, the meaning of the parameter must be documented with the group. An example of this might be documented the sync points in a multiplex of several video streams; the group definition might be ‘Is an I frame’, and the group parameter might be the identifier of each stream. Since the sample to group box occurs once for each stream, it is now both compact, and informs the reader about each stream separately.

One example of using these tables is to represent the assignments of samples to *layers.* In this case each sample group represents one layer, with an instance of the `SampleToGroup` box describing which layer a sample belongs to.

8.9.2 Sample to Group Box

8.9.2.1 Definition

Box Type: `'sbgp'`
Container: Sample Table Box (`'stbl'`) or Track Fragment Box (`'traf'`)
Mandatory: No
Quantity: Zero or more.

This table can be used to find the group that a sample belongs to and the associated description of that sample group. The table is compactly coded with each entry giving the index of the first sample of a run of samples with the same sample group descriptor. The sample group description ID is an index that refers to a `SampleGroupDescription` box, which contains entries describing the characteristics of each sample group.

There may be multiple instances of this box if there is more than one sample grouping for the samples in a track. Each instance of the `SampleToGroup` box has a type code that distinguishes different sample groupings. Within a track, there shall be at most one instance of this box with a particular grouping type. The associated `SampleGroupDescription` shall indicate the same value for the grouping type.

Version 1 of this box should only be used if a grouping type parameter is needed.

8.9.2.2 Syntax

```
aligned(8) class SampleToGroupBox
   extends FullBox('sbgp', version, 0)
{
   unsigned int(32)  grouping_type;
   if (version == 1) {
      unsigned int(32) grouping_type_parameter;
   }
   unsigned int(32)  entry_count;
   for (i=1; i <= entry_count; i++)
   {
      unsigned int(32)  sample_count;
      unsigned int(32)  group_description_index;
   }
}
```

8.9.2.3 Semantics

`version` is an integer that specifies the version of this box, either 0 or 1.
`grouping_type` is an integer that identifies the type (i.e. criterion used to form the sample groups) of the sample grouping and links it to its sample group description table with the same value for grouping type. At most one occurrence of this box with the same value for `grouping_type` (and, if used, `grouping_type_parameter`) shall exist for a track.
`grouping_type_parameter` is an indication of the sub-type of the grouping
`entry_count` is an integer that gives the number of entries in the following table.
`sample_count` is an integer that gives the number of consecutive samples with the same sample group descriptor. If the sum of the sample count in this box is less than the total sample count, then the reader should effectively extend it with an entry that associates the remaining samples with no group. It is an error for the total in this box to be greater than the `sample_count` documented elsewhere, and the reader behaviour would then be undefined.

`group_description_index` is an integer that gives the index of the sample group entry which describes the samples in this group. The index ranges from 1 to the number of sample group entries in the `SampleGroupDescription` Box, or takes the value 0 to indicate that this sample is a member of no group of this type.

8.9.3 Sample Group Description Box

8.9.3.1 Definition

Box Type: `'sgpd'`
Container: Sample Table Box (`'stbl'`) or Track Fragment Box (`'traf'`)
Mandatory: No
Quantity: Zero or more, with one for each Sample to Group Box.

This description table gives information about the characteristics of sample groups. The descriptive information is any other information needed to define or characterize the sample group.

There may be multiple instances of this box if there is more than one sample grouping for the samples in a track. Each instance of the `SampleGroupDescription` box has a type code that distinguishes different sample groupings. Within a track, there shall be at most one instance of this box with a particular grouping type. The associated `SampleToGroup` shall indicate the same value for the grouping type.

The information is stored in the sample group description box after the entry-count. An abstract entry type is defined and sample groupings shall define derived types to represent the description of each sample group. For video tracks, an abstract `VisualSampleGroupEntry` is used with similar types for audio and hint tracks.

NOTE In version 0 of the entries the base classes for sample group description entries are neither boxes nor have a size is signaled. For this reason, use of version 0 entries is deprecated. When defining derived classes, ensure either that they have a fixed size, or that the size is explicitly indicated with a length field. An implied size (e.g. achieved by parsing the data) is not recommended as this makes scanning the array difficult.

8.9.3.2 Syntax

```
// Sequence Entry
abstract class SampleGroupDescriptionEntry (unsigned int(32) grouping_type)
{
}

abstract class VisualSampleGroupEntry (unsigned int(32) grouping_type) extends
SampleGroupDescriptionEntry (grouping_type)
{
}

abstract class AudioSampleGroupEntry (unsigned int(32) grouping_type) extends
SampleGroupDescriptionEntry (grouping_type)
{
}

abstract class HintSampleGroupEntry (unsigned int(32) grouping_type) extends
SampleGroupDescriptionEntry (grouping_type)
{
}
```

```
aligned(8) class SampleGroupDescriptionBox (unsigned int(32) handler_type)
   extends FullBox('sgpd', version, 0){
   unsigned int(32) grouping_type;
   if (version==1) { unsigned int(32) default_length; }
   unsigned int(32) entry_count;
   int i;
   for (i = 1 ; i <= entry_count ; i++){
      if (version==1) {
         if (default_length==0) {
            unsigned int(32) description_length;
         }
      }
      switch (handler_type){
         case 'vide': // for video tracks
            VisualSampleGroupEntry (grouping_type);
            break;
         case 'soun': // for audio tracks
            AudioSampleGroupEntry(grouping_type);
            break;
         case 'hint': // for hint tracks
            HintSampleGroupEntry(grouping_type);
            break;
      }
   }
}
```

8.9.3.3 Semantics

`version` is an integer that specifies the version of this box.
`grouping_type` is an integer that identifies the `SampleToGroup` box that is associated with this sample group description.
`entry_count` is an integer that gives the number of entries in the following table.
`default_length` indicates the length of every group entry (if the length is constant), or zero (0) if it is variable
`description_length` indicates the length of an individual group entry, in the case it varies from entry to entry and `default_length` is therefore 0

8.9.4 Representation of group structures in Movie Fragments

Support for Sample Group structures within Movie fragments is provided by the use of the `SampleToGroup` Box with the container for this Box being the `Track Fragment` Box (`'traf'`). The definition, syntax and semantics of this Box is as specified in subclause 8.9.2.

The `SampleToGroup` Box can be used to find the group that a sample in a track fragment belongs to and the associated description of that sample group. The table is compactly coded with each entry giving the index of the first sample of a run of samples with the same sample group descriptor. The sample group description ID is an index that refers to a `SampleGroupDescription` Box, which contains entries describing the characteristics of each sample group and present in the `SampleTableBox`.

There may be multiple instances of the `SampleToGroup` Box if there is more the one sample grouping for the samples in a track fragment. Each instance of the `SampleToGroup` Box has a type code that distinguishes different sample groupings. The associated `SampleGroupDescription` shall indicate the same value for the grouping type.

The total number of samples represented in any `SampleToGroup` Box in the track fragment must match the total number of samples in all the track fragment runs. Each `SampleToGroup` Box documents a different grouping of the same samples.

Zero or more SampleGroupDescription boxes may also be present in a Track Fragment Box. These definitions are additional to the definitions provided in the Sample Table of the track in the Movie Box. Group definitions within a movie fragment can also be referenced and used from within that same movie fragment.

Within the SampleToGroup box in that movie fragment, the group description indexes for groups defined within the same fragment start at 0x10001, i.e. the index value 1, with the value 1 in the top 16 bits. This means there must be fewer than 65536 group definitions for this track and grouping type in the sample table in the Movie Box.

When changing the size of movie fragments, or removing them, these fragment-local group definitions will need to be merged into the definitions in the movie box, or into the new movie fragments, and the index numbers in the SampleToGroup box(es) adjusted accordingly. It is recommended that, in this process, identical (and hence duplicate) definitions not be made in any SampleGroupDescription box, but that duplicates be merged and the indexes adjusted accordingly.

8.10 User Data

8.10.1 User Data Box

8.10.1.1 Definition

Box Type: `'udta'`
Container: Movie Box (`'moov'`) or Track Box (`'trak'`)
Mandatory: No
Quantity: Zero or one

This box contains objects that declare user information about the containing box and its data (presentation or track).

The User Data Box is a container box for informative user-data. This user data is formatted as a set of boxes with more specific box types, which declare more precisely their content.

Only a copyright notice is defined in this specification.

8.10.1.2 Syntax

```
aligned(8) class UserDataBox extends Box('udta') {
}
```

8.10.2 Copyright Box

8.10.2.1 Definition

Box Type: `'cprt'`
Container: User data box (`'udta'`)
Mandatory: No
Quantity: Zero or more

The Copyright box contains a copyright declaration which applies to the entire presentation, when contained within the Movie Box, or, when contained in a track, to that entire track. There may be multiple copyright boxes using different language codes.

8.10.2.2 Syntax

```
aligned(8) class CopyrightBox
   extends FullBox('cprt', version = 0, 0) {
   const bit(1)   pad = 0;
   unsigned int(5)[3]   language;   // ISO-639-2/T language code
   string   notice;
}
```

8.10.2.3 Semantics

`language` declares the language code for the following text. 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. The code is confined to being three lower-case letters, so these values are strictly positive.

`notice` is a null-terminated string in either UTF-8 or UTF-16 characters, giving a copyright notice. If UTF-16 is used, the string shall start with the BYTE ORDER MARK (0xFEFF), to distinguish it from a UTF-8 string. This mark does not form part of the final string.

8.10.3 Track Selection Box

8.10.3.1 Introduction

A typical presentation stored in a file contains one alternate group per media type: one for video, one for audio, etc. Such a file may include several video tracks, although, at any point in time, only one of them should be played or streamed. This is achieved by assigning all video tracks to the same alternate group. (See subclause 8.3.2 for the definition of alternate groups.)

All tracks in an alternate group are candidates for media selection, but it may not make sense to switch between some of those tracks during a session. One may for instance allow switching between video tracks at different bitrates and keep frame size but not allow switching between tracks of different frame size. In the same manner it may be desirable to enable selection – but not switching – between tracks of different video codecs or different audio languages.

The distinction between tracks for *selection* and *switching* is addressed by assigning tracks to switch groups in addition to alternate groups. One alternate group may contain one or more switch groups. All tracks in an alternate group are candidates for media selection, while tracks in a switch group are also available for switching during a session. Different switch groups represent different operation points, such as different frame size, high/low quality, etc.

For the case of non-scalable bitstreams, several tracks may be included in a switch group. The same also applies to non-layered scalable bitstreams, such as traditional AVC streams.

By labelling tracks with attributes it is possible to characterize them. Each track can be labelled with a list of attributes which can be used to describe tracks in a particular switch group or differentiate tracks that belong to different switch groups.

8.10.3.2 Definition

Box Type: `'tsel'`
Container: User Data Box (`'udta'`)
Mandatory: No
Quantity: Zero or One

The track selection box is contained in the user data box of the track it modifies.

8.10.3.3 Syntax

```
aligned(8) class TrackSelectionBox
   extends FullBox('tsel', version = 0, 0) {
   template int(32) switch_group = 0;
   unsigned int(32) attribute_list[];      // to end of the box
}
```

8.10.3.4 Semantics

`switch_group` is an integer that specifies a group or collection of tracks. If this field is 0 (default value) or if the Track Selection box is absent there is no information on whether the track can be used for switching during playing or streaming. If this integer is not 0 it shall be the same for tracks that can be used for switching between each other. Tracks that belong to the same switch group shall belong to the same alternate group. A switch group may have only one member.

`attribute_list` is a list, to the end of the box, of attributes. The attributes in this list should be used as descriptions of tracks or differentiation criteria for tracks in the same alternate or switch group. Each differentiating attribute is associated with a pointer to the field or information that distinguishes the track.

8.10.3.5 Attributes

The following attributes are descriptive:

Name	*Attribute*	*Description*
Temporal scalability	`'tesc'`	The track can be temporally scaled.
Fine-grain SNR scalability	`'fgsc'`	The track can be scaled in terms of quality.
Coarse-grain SNR scalability	`'cgsc'`	The track can be scaled in terms of quality.
Spatial scalability	`'spsc'`	The track can be spatially scaled.
Region-of-interest scalability	`'resc'`	The track can be region-of-interest scaled.
View scalability	`'vwsc'`	The track can be scaled in terms of number of views.

The following attributes are differentiating:

Name	*Attribute*	*Pointer*
Codec	`'cdec'`	Sample Entry (in Sample Description box of media track)
Screen size	`'scsz'`	Width and height fields of Visual Sample Entries.
Max packet size	`'mpsz'`	`Maxpacketsize` field in RTP Hint Sample Entry
Media type	`'mtyp'`	`Handlertype` in Handler box (of media track)
Media language	`'mela'`	Language field in Media Header box
Bitrate	`'bitr'`	Total size of the samples in the track divided by the duration in the track header box
Frame rate	`'frar'`	Number of samples in the track divided by duration in the track header box
Number of views	`'nvws'`	Number of views in the sub track

Descriptive attributes characterize the tracks they modify, whereas differentiating attributes differentiate between tracks that belong to the same alternate or switch groups. The pointer of a differentiating attribute indicates the location of the information that differentiates the track from other tracks with the same attribute.

8.11 Metadata Support

A common base structure is used to contain general metadata, called the meta box.

8.11.1 The Meta box

8.11.1.1 Definition

Box Type: 'meta'
Container: File, Movie Box ('moov'), Track Box ('trak'), or Additional Metadata Container Box ('meco')
Mandatory: No
Quantity: Zero or one (in File, 'moov', and 'trak'), One or more (in 'meco')

A meta box contains descriptive or annotative metadata. The 'meta' box is required to contain a 'hdlr' box indicating the structure or format of the 'meta' box contents. That metadata is located either within a box within this box (e.g. an XML box), or is located by the item identified by a primary item box.

All other contained boxes are specific to the format specified by the handler box.

The other boxes defined here may be defined as optional or mandatory for a given format. If they are used, then they must take the form specified here. These optional boxes include a data-information box, which documents other files in which metadata values (e.g. pictures) are placed, and a item location box, which documents where in those files each item is located (e.g. in the common case of multiple pictures stored in the same file). At most one meta box may occur at each of the file level, movie level, or track level, unless they are contained in an additional metadata container box ('meco').

If an Item Protection Box occurs, then some or all of the meta-data, including possibly the primary resource, may have been protected and be un-readable unless the protection system is taken into account.

8.11.1.2 Syntax

```
aligned(8) class MetaBox (handler_type)
   extends FullBox('meta', version = 0, 0) {
   HandlerBox(handler_type)    theHandler;
   PrimaryItemBox       primary_resource;      // optional
   DataInformationBox   file_locations;        // optional
   ItemLocationBox      item_locations;        // optional
   ItemProtectionBox    protections;           // optional
   ItemInfoBox          item_infos;            // optional
   IPMPControlBox       IPMP_control;          // optional
   ItemReferenceBox     item_refs;             // optional
   ItemDataBox          item_data;             // optional
   Box   other_boxes[];                        // optional
}
```

8.11.1.3 Semantics

The structure or format of the metadata is declared by the handler. In the case that the primary data is identified by a primary item, and that primary item has an item information entry with an item_type, the handler type may be the same as the item_type.

8.11.2 XML Boxes

8.11.2.1 Definition

Box Type: `'xml '` or `'bxml'`
Container: Meta box (`'meta'`)
Mandatory: No
Quantity: Zero or one

When the primary data is in XML format and it is desired that the XML be stored directly in the meta-box, one of these forms may be used. The Binary XML Box may only be used when there is a single well-defined binarization of the XML for that defined format as identified by the handler.

Within an XML box the data is in UTF-8 format unless the data starts with a byte-order-mark (BOM), which indicates that the data is in UTF-16 format.

8.11.2.2 Syntax

```
aligned(8) class XMLBox
      extends FullBox('xml ', version = 0, 0) {
   string xml;
}

aligned(8) class BinaryXMLBox
      extends FullBox('bxml', version = 0, 0) {
   unsigned int(8) data[];     // to end of box
}
```

8.11.3 The Item Location Box

8.11.3.1 Definition

Box Type: `'iloc'`
Container: Meta box (`'meta'`)
Mandatory: No
Quantity: Zero or one

The item location box provides a directory of resources in this or other files, by locating their containing file, their offset within that file, and their length. Placing this in binary format enables common handling of this data, even by systems which do not understand the particular metadata system (handler) used. For example, a system might integrate all the externally referenced metadata resources into one file, re-adjusting file offsets and file references accordingly.

The box starts with three or four values, specifying the size in bytes of the `offset` field, `length` field, `base_offset` field, and, in version 1 of this box, the `extent_index` fields, respectively. These values must be from the set {0, 4, 8}.

The `construction_method` field indicates the 'construction method' for the item:

i) file_offset: by the usual absolute file offsets into the file at data_reference_index; (construction_method == 0)

ii) idat_offset: by box offsets into the idat box in the same meta box; neither the data_reference_index nor extent_index fields are used; (construction_method == 1)

iii) item_offset: by item offset into the items indicated by a new extent_index field, which is only used (currently) by this construction method. (construction_method == 2).

The extent_index is only used for the method item_offset; it indicates the 1-based index of the item reference with referenceType 'iloc' linked from this item. If index_size is 0, then the value 1 is implied; the value 0 is reserved.

Items may be stored fragmented into extents, e.g. to enable interleaving. An extent is a contiguous subset of the bytes of the resource; the resource is formed by concatenating the extents. If only one extent is used (`extent_count` = 1) then either or both of the offset and length may be implied:

- If the offset is not identified (the field has a length of zero), then the beginning of the source (offset 0 into the file, idat box, or other item) is implied.
- If the length is not specified, or specified as zero, then the entire length of the source is implied. References into the same file as this metadata, or items divided into more than one extent, should have an explicit offset and length, or use a MIME type requiring a different interpretation of the file, to avoid infinite recursion.

The size of the item is the sum of the extent lengths.

NOTE Extents may be interleaved with the chunks defined by the sample tables of tracks.

The data-reference index may take the value 0, indicating a reference into the same file as this metadata, or an index into the data-reference table.

Some referenced data may itself use offset/length techniques to address resources within it (e.g. an MP4 file might be 'included' in this way). Normally such offsets are relative to the beginning of the containing file. The field 'base offset' provides an additional offset for offset calculations within that contained data. For example, if an MP4 file is included within a file formatted to this specification, then normally data-offsets within that MP4 section are relative to the beginning of file; the base offset adds to those offsets.

If an item is constructed from other items, and those source items are protected, the offset and length information apply to the source items after they have been de-protected. That is, the target item data is formed from unprotected source data.

For maximum compatibility, version 0 of this box should be used in preference to version 1 with construction_method==0, when possible.

8.11.3.2 Syntax

```
aligned(8) class ItemLocationBox extends FullBox('iloc', version, 0) {
   unsigned int(4)   offset_size;
   unsigned int(4)   length_size;
   unsigned int(4)   base_offset_size;
   if (version == 1)
      unsigned int(4)   index_size;
   else
      unsigned int(4)   reserved;
   unsigned int(16)  item_count;
   for (i=0; i<item_count; i++) {
      unsigned int(16)  item_ID;
      if (version == 1) {
         unsigned int(12)  reserved = 0;
         unsigned int(4)   construction_method;
      }
      unsigned int(16)  data_reference_index;
      unsigned int(base_offset_size*8)  base_offset;
      unsigned int(16)     extent_count;
      for (j=0; j<extent_count; j++) {
         if ((version == 1) && (index_size > 0)) {
            unsigned int(index_size*8)  extent_index;
         }
         unsigned int(offset_size*8) extent_offset;
```

```
            unsigned int(length_size*8) extent_length;
        }
    }
}
```

8.11.3.3 Semantics

`offset_size` is taken from the set {0, 4, 8} and indicates the length in bytes of the `offset` field.
`length_size` is taken from the set {0, 4, 8} and indicates the length in bytes of the `length` field.
`base_offset_size` is taken from the set {0, 4, 8} and indicates the length in bytes of the `base_offset` field.
`index_size` is taken from the set {0, 4, 8} and indicates the length in bytes of the `extent_index` field.
`item_count` counts the number of resources in the following array.
`item_ID` is an arbitrary integer 'name' for this resource which can be used to refer to it (e.g. in a URL).
`construction_method` is taken from the set 0 (file), 1 (idat) or 2 (item)
`data-reference-index` is either zero ('this file') or a 1-based index into the data references in the data information box.
`base_offset` provides a base value for offset calculations within the referenced data. If `base_offset_size` is 0, `base_offset` takes the value 0, i.e. it is unused.
`extent_count` provides the count of the number of extents into which the resource is fragmented; it must have the value 1 or greater
`extent_index` provides an index as defined for the construction method
`extent_offset` provides the absolute offset in bytes from the beginning of the containing file, of this item. If `offset_size` is 0, `offset` takes the value 0
`extent_length` provides the absolute length in bytes of this metadata item. If `length_size` is 0, `length` takes the value 0. If the value is 0, then length of the item is the length of the entire referenced file.

8.11.4 Primary Item Box

8.11.4.1 Definition

Box Type: `'pitm'`
Container: Meta box (`'meta'`)
Mandatory: No
Quantity: Zero or one

For a given handler, the primary data may be one of the referenced items when it is desired that it be stored elsewhere, or divided into extents; or the primary metadata may be contained in the meta-box (e.g. in an XML box). Either this box must occur, or there must be a box within the meta-box (e.g. an XML box) containing the primary information in the format required by the identified handler.

8.11.4.2 Syntax

```
aligned(8) class PrimaryItemBox
      extends FullBox('pitm', version = 0, 0) {
   unsigned int(16)  item_ID;
}
```

8.11.4.3 Semantics

`item_ID` is the identifier of the primary item

8.11.5 Item Protection Box

8.11.5.1 Definition

Box Type: 'ipro'
Container: Meta box ('meta')
Mandatory: No
Quantity: Zero or one

The item protection box provides an array of item protection information, for use by the Item Information Box.

8.11.5.2 Syntax

```
aligned(8) class ItemProtectionBox
      extends FullBox('ipro', version = 0, 0) {
   unsigned int(16) protection_count;
   for (i=1; i<=protection_count; i++) {
      ProtectionSchemeInfoBox protection_information;
   }
}
```

8.11.6 Item Information Box

8.11.6.1 Definition

Box Type: 'iinf'
Container: Meta Box ('meta')
Mandatory: No
Quantity: Zero or one

The Item information box provides extra information about selected items, including symbolic ('file') names. It may optionally occur, but if it does, it must be interpreted, as item protection or content encoding may have changed the format of the data in the item. If both content encoding and protection are indicated for an item, a reader should first un-protect the item, and then decode the item's content encoding. If more control is needed, an IPMP sequence code may be used.

This box contains an array of entries, and each entry is formatted as a box. This array is sorted by increasing `item_ID` in the entry records.

Three versions of the item info entry are defined. Version 1 includes additional information to version 0 as specified by an extension type. For instance, it shall be used with extension type `'fdel'` for items that are referenced by the file partition box (`'fpar'`), which is defined for source file partitionings and applies to file delivery transmissions. Version 2 provides an alternative structure in which metadata item types are indicated by a 32-bit (typically 4-character) registered or defined code; two of these codes are defined to indicate a MIME type or metadata typed by a URI.

If no extension is desired, the box may terminate without the `extension_type` field and the extension; if, in addition, `content_encoding` is not desired, that field also may be absent and the box terminate before it. If an extension is desired without an explicit `content_encoding`, a single null byte, signifying the empty string, must be supplied for the `content_encoding`, before the indication of `extension_type`.

If file delivery item information is needed and a version 2 ItemInfoEntry is used, then the file delivery information is stored (a) as a separate item of type 'fdel') (b) linked by an item reference from the item, to the file delivery information, of type 'fdel'. There must be exactly one such reference if file delivery information is needed.

It is possible that there are valid URI forms for MPEG-7 metadata (e.g. a schema URI with a fragment identifying a particular element), and it may be possible that these structures could be used for MPEG-7. However, there is explicit support for MPEG-7 in ISO base media file format family files, and this explicit support is preferred as it allows, among other things:

a) incremental update of the metadata (logically, I/P coding, in video terms) whereas this draft is 'I-frame only';

b) binarization and thus compaction;

c) the use of multiple schemas.

Therefore, the use of these structures for MPEG-7 is deprecated (and undocumented).

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

8.11.6.2 Syntax

```
aligned(8) class ItemInfoExtension(unsigned int(32) extension_type)
{
}
aligned(8) class FDItemInfoExtension() extends ItemInfoExtension ('fdel')
{
   string            content_location;
   string            content_MD5;
   unsigned int(64)  content_length;
   unsigned int(64)  transfer_length;
   unsigned int(8)   entry_count;
   for (i=1; i <= entry_count; i++)
      unsigned int(32)  group_id;
}
aligned(8) class ItemInfoEntry
      extends FullBox('infe', version, 0) {
   if ((version == 0) || (version == 1)) {
      unsigned int(16)  item_ID;
      unsigned int(16)  item_protection_index
      string            item_name;
      string            content_type;
      string            content_encoding;   //optional
   }
   if (version == 1) {
      unsigned int(32)  extension_type;     //optional
      ItemInfoExtension(extension_type);    //optional
   }
   if (version == 2) {
      unsigned int(16)  item_ID;
      unsigned int(16)  item_protection_index;
      unsigned int(32)  item_type;

      string            item_name;
      if (item_type=='mime') {
         string         content_type;
         string         content_encoding;   //optional
      } else if (item_type == 'uri ') {
         string         item_uri_type;
      }
   }
}
aligned(8) class ItemInfoBox
      extends FullBox('iinf', version = 0, 0) {
   unsigned int(16) entry_count;
   ItemInfoEntry[ entry_count ]     item_infos;
}
```

8.11.6.3 Semantics

`item_id` contains either 0 for the primary resource (e.g., the XML contained in an `'xml '` box) or the ID of the item for which the following information is defined.
`item_protection_index` contains either 0 for an unprotected item, or the one-based index into the item protection box defining the protection applied to this item (the first box in the item protection box has the index 1).
`item_name` is a null-terminated string in UTF-8 characters containing a symbolic name of the item (source file for file delivery transmissions).
`item_type` is a 32-bit value, typically 4 printable characters, that is a defined valid item type indicator, such as 'mime'
`content_type` is a null-terminated string in UTF-8 characters with the MIME type of the item. If the item is content encoded (see below), then the content type refers to the item after content decoding.
`item_uri_type` is a string that is an absolute URI, that is used as a type indicator.
`content_encoding` is an optional null-terminated string in UTF-8 characters used to indicate that the binary file is encoded and needs to be decoded before interpreted. The values are as defined for Content-Encoding for HTTP/1.1. Some possible values are "gzip", "compress" and "deflate". An empty string indicates no content encoding. Note that the item is stored after the content encoding has been applied.
`extension_type` is a printable four-character code that identifies the extension fields of version 1 with respect to version 0 of the Item information entry.
`content_location` is a null-terminated string in UTF-8 characters containing the URI of the file as defined in HTTP/1.1 (RFC 2616).
`content_MD5` is a null-terminated string in UTF-8 characters containing an MD5 digest of the file. See HTTP/1.1 (RFC 2616) and RFC 1864.
`content_length` gives the total length (in bytes) of the (un-encoded) file.
`transfer_length` gives the total length (in bytes) of the (encoded) file. Note that transfer length is equal to content length if no content encoding is applied (see above).
`entry_count` provides a count of the number of entries in the following array.
`group_ID` indicates a file group to which the file item (source file) belongs. See 3GPP TS 26.346 for more details on file groups.

8.11.7 Additional Metadata Container Box

8.11.7.1 Definition

Box Type: `'meco'`
Container: File, Movie Box (`'moov'`), or Track Box (`'trak'`)
Mandatory: No
Quantity: Zero or one

The additional metadata container box includes one or more meta boxes. It can be carried at the top level of the file, in the Movie Box (`'moov'`), or in the Track Box (`'trak'`) and shall only be present if it is accompanied by a meta box in the same container. A meta box that is not contained in the additional metadata container box is the preferred (primary) meta box. Meta boxes in the additional metadata container box complement or give alternative metadata information. The usage of multiple meta boxes may be desirable when, e.g., a single handler is not capable of processing all metadata. All meta boxes at a certain level, including the preferred one and those contained in the additional metadata container box, must have different handler types.

A meta box contained in an additional metadata container box shall contain a primary Item box or the primary data box required by the handler (e.g., an XML Box). It shall not include boxes or syntax elements concerning items other than the primary item indicated by the present primary item box or XML box. URLs in a meta box contained in an additional metadata container box are relative to the context of the preferred meta box.

8.11.7.2 Syntax

```
aligned(8) class AdditionalMetadataContainerBox extends Box('meco') {
}
```

8.11.8 Metabox Relation Box

8.11.8.1 Definition

Box Type: 'mere'
Container: Additional Metadata Container Box ('meco')
Mandatory: No
Quantity: Zero or more

The metabox relation box indicates a relation between two meta boxes at the same level, i.e. the top level of the file, the Movie Box, or Track Box. The relation between two meta boxes is unspecified if there is no metabox relation box for those meta boxes. Meta boxes are referenced by specifying their handler types.

8.11.8.2 Syntax

```
aligned(8) class MetaboxRelationBox
   extends FullBox('mere', version=0, 0) {
   unsigned int(32)  first_metabox_handler_type;
   unsigned int(32)  second_metabox_handler_type;
   unsigned int(8)   metabox_relation;
}
```

8.11.8.3 Semantics

`first_metabox_handler_type` indicates the first meta box to be related.
`second_metabox_handler_type` indicates the second meta box to be related.
`metabox_relation` indicates the relation between the two meta boxes. The following values are defined:

1 The relationship between the boxes is unknown (which is the default when this box is not present);

2 the two boxes are semantically un-related (e.g., one is presentation, the other annotation);

3 the two boxes are semantically related but complementary (e.g., two disjoint sets of meta-data expressed in two different meta-data systems);

4 the two boxes are semantically related but overlap (e.g., two sets of meta-data neither of which is a subset of the other); neither is 'preferred' to the other;

5 the two boxes are semantically related but the second is a proper subset or weaker version of the first; the first is preferred;

6 the two boxes are semantically related and equivalent (e.g., two essentially identical sets of meta-data expressed in two different meta-data systems).

8.11.9 URL Forms for meta boxes

When a meta-box is used, then URLs may be used to refer to items in the meta-box, either using an absolute URL, or using a relative URL. Absolute URLs may only be used to refer to items in a file-level meta box.

When interpreting data that is in the context of a meta-box (i.e. the file for a file-level meta-box, the presentation for a movie-level meta-box, or the track for a track-level meta-box), the items in the meta-box are treated as *shadowing* files in the same location as that from which the container file came. This shadowing means that a reference to another file in the same location as the container file may be resolved to an item within the container file itself. Items can be addressed within the container file by appending a fragment to the URL for the container file itself. That fragment starts with the "#" character and consists of either:

a) `item_ID=<n>`, identifying the item by its ID (the ID may be 0 for the primary resource);

b) `item_name=<item_name>`, when the item information box is used.

If a fragment within the contained item must be addressed, then the initial "#" character of that fragment is replaced by "*".

Consider the following example: `<http://a.com/d/v.qrv#item_name=tree.html*branch1>`. We assume that `v.qrv` is a file with a meta-box at the file level. First, the client strips the fragment and fetches `v.qrv` from a.com using HTTP. It then inspects the top-level meta box and adds the items in it, logically, to its cache of the directory "d" on a.com. It then re-forms the URL as `<http://a.com/d/tree.html#branch1>`. Note that the fragment has been elevated to a full file name, and the first "*" has been transformed back into a "#". The client then either finds an item named `tree.html` in the meta box, or fetches `tree.html` from `a.com`, and it then finds the anchor "`branch1`" within `tree.html`. If within that html, a file was referenced using a relative URL, e.g. "`flower.gif`", then the client converts this to an absolute URL using the normal rules: `<http://a.com/d/flower.gif>` and again it checks to see if `flower.gif` is a named item (and hence shadowing a separate file of this name), and then if it is not, fetches `flower.gif` from `a.com`.

8.11.10 Static Metadata

This section defines the storage of static (un-timed) metadata in the ISO file format family.

Reader support for metadata in general is optional, and therefore it is also optional for the formats defined here or elsewhere, unless made mandatory by a derived specification.

8.11.10.1 Simple textual

There is existing support for simple textual tags in the form of the user-data boxes; currently only one is defined – the copyright notice. Other metadata is permitted using this simple form if:

a) it uses a registered box-type or it uses the UUID escape (the latter is permitted today);

b) it uses a registered tag, the equivalent MPEG-7 construct must be documented as part of the registration.

8.11.10.2 Other forms

When other forms of metadata are desired, then a `'meta'` box as defined above may be included at the appropriate level of the document. If the document is intended to be primarily a metadata document per se, then the meta box is at file level. If the metadata annotates an entire presentation, then the meta box is at the movie level; an entire stream, at the track level.

8.11.10.3 MPEG-7 metadata

MPEG-7 metadata is stored in meta boxes to this specification.

1) The handler-type is `'mp7t'` for textual metadata in Unicode format;

2) The handler-type is `'mp7b'` for binary metadata compressed in the BIM format. In this case, the binary XML box contains the configuration information immediately followed by the binarized XML.

3) When the format is textual, there is either another box in the metadata container `'meta'`, called `'xml'`, which contains the textual MPEG-7 document, or there is a primary item box identifying the item containing the MPEG-7 XML.

4) When the format is binary, there is either another box in the metadata container `'meta'`, called `'bxml'`, which contains the binary MPEG-7 document, or a primary item box identifying the item containing the MPEG-7 binarized XML.

5) If an MPEG-7 box is used at the file level, then the brand `'mp71'` should be a member of the compatible-brands list in the file-type box.

8.11.11 Item Data Box

8.11.11.1 Definition

Box Type: `'idat'`
Container: Metadata box (`'meta'`)
Mandatory: No
Quantity: Zero or one

This box contains the data of metadata items that use the construction method indicating that an item's data extents are stored within this box.

8.11.11.2 Syntax

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

8.11.11.3 Semantics

`data` is the contained meta data

8.11.12 Item Reference Box

8.11.12.1 Definition

Box Type: `'iref'`
Container: Metadata box (`'meta'`)
Mandatory: No
Quantity: Zero or one

The item reference box allows the linking of one item to others via typed references. All the references for one item of a specific type are collected into a single item type reference box, whose type is the reference type, and which has a 'from item ID' field indicating which item is linked. The items linked to are then represented by an array of 'to item ID's. All these single item type reference boxes are then collected into the item reference box. The reference types defined for the track reference box defined in 8.3.3 may be used here if appropriate, or other registered reference types.

NOTE: This design makes it fairly easy to find all the references of a specific type, or from a specific item.

8.11.12.2 Syntax

```
aligned(8) class SingleItemTypeReferenceBox(referenceType) extends
Box(referenceType) {
   unsigned int(16) from_item_ID;
   unsigned int(16) reference_count;
   for (j=0; j<reference_count; j++) {
      unsigned int(16) to_item_ID;
   }
}
aligned(8) class ItemReferenceBox extends FullBox('iref', version=0, 0) {
   SingleItemTypeReferenceBox references[];
}
```

8.11.12.3 Semantics

`reference_type` contains an indication of the type of the reference
`from_item_id` contains the ID of the item that refers to other items
`reference_count` is the number of references
`to_item_id` contains the ID of the item referred to

8.11.13 Auxiliary video metadata

An auxiliary video track used for depth or parallax information may carry a meta-data item of type 'auvd' (auxiliary video descriptor); the data of that item is exactly one si_rbsp() as specified in ISO/IEC 23002-3. (Note that si_rbsp() is externally framed, and the length is supplied by the item location information in the file format). There may be more than one of these meta-data items (e.g. one for parallax info and one for depth, in the case that the same stream serves).

8.12 Support for Protected Streams

This section documents the file-format transformations which are used for protected content. These transformations can be used under several circumstances:

- They must be used when the content has been transformed (e.g. by encryption) in such a way that it can no longer be decoded by the normal decoder;
- They may be used when the content should only be decoded when the protection system is understood and implemented.

The transformation functions by *encapsulating* the original media declarations. The encapsulation changes the four-character-code of the sample entries, so that protection-unaware readers see the media stream as a new stream format.

Because the format of a sample entry varies with media-type, a different encapsulating four-character-code is used for each media type (audio, video, text etc.). They are:

Stream (Track) Type	Sample-Entry Code
Video	`encv`
Audio	`enca`
Text	`enct`
System	`encs`

The transformation of the sample description is described by the following procedure:

1) The four-character-code of the sample description is replaced with a four-character-code indicating protection encapsulation: these codes vary only by media-type. For example, `'mp4v'` is replaced with `'encv'` and `'mp4a'` is replaced with `'enca'`.

2) A `ProtectionSchemeInfoBox` (*defined below*) is added to the sample description, leaving all other boxes unmodified.

3) The original sample entry type (four-character-code) is stored within the `ProtectionSchemeInfoBox`, in a new box called the `OriginalFormatBox` (*defined below*);

There are then three methods for signalling the nature of the protection, which may be used individually or in combination.

1) When MPEG-4 systems is used, then IPMP *must* be used to signal that the streams are protected.

2) IPMP descriptors may also be used outside the MPEG-4 systems context using boxes containing IPMP descriptors.

3) The protection applied may also be described using the scheme type and information boxes.

When IPMP is used outside of MPEG-4 systems, then a 'global' `IPMPControlBox` may also occur within the `'moov'` atom.

NOTE When MPEG-4 systems is used, an MPEG-4 systems terminal can effectively treat, for example, `'encv'` with an Original Format of `'mp4v'` exactly the same as `'mp4v'`, by using the IPMP descriptors.

8.12.1 Protection Scheme Information Box

8.12.1.1 Definition

Box Types: `'sinf'`
Container: Protected Sample Entry, or Item Protection Box (`'ipro'`)
Mandatory: Yes
Quantity: ~~Exactly one~~One or More

The Protection Scheme Information Box contains all the information required both to understand the encryption transform applied and its parameters, and also to find other information such as the kind and location of the key management system. It also documents the original (unencrypted) format of the media. The Protection Scheme Information Box is a container Box. It is mandatory in a sample entry that uses a code indicating a protected stream.

When used in a protected sample entry, this box must contain the original format box to document the original format. At least one of the following signalling methods must be used to identify the protection applied:

a) MPEG-4 systems with IPMP: no other boxes, when IPMP descriptors in MPEG-4 systems streams are used;

b) Scheme signalling: a `SchemeTypeBox` and `SchemeInformationBox`, when these are used (either both must occur, or neither).

At least one protection scheme information box must occur in a protected sample entry. When more than one occurs, they are equivalent, alternative, descriptions of the same protection. Readers should choose one to process.

8.12.1.2 Syntax

```
aligned(8) class ProtectionSchemeInfoBox(fmt) extends Box('sinf') {
   OriginalFormatBox(fmt)  original_format;

   SchemeTypeBox           scheme_type_box;    // optional
   SchemeInformationBox    info;         // optional
}
```

8.12.2 Original Format Box

8.12.2.1 Definition

Box Types: `'frma'`
Container: Protection Scheme Information Box (`'sinf'`) or Restricted Scheme Information Box (`'rinf'`)
Mandatory: Yes when used in a protected sample entry or in a restricted sample entry
Quantity: Exactly one

The Original Format Box `'frma'` contains the four-character-code of the original un-transformed sample description:

8.12.2.2 Syntax

```
aligned(8) class OriginalFormatBox(codingname) extends Box ('frma') {
   unsigned int(32)  data_format = codingname;
                     // format of decrypted, encoded data (in case of protection)
                     // or un-transformed sample entry (in case of restriction)
}
```

8.12.2.3 Semantics

`data_format` is the four-character-code of the original un-transformed sample entry (e.g. “mp4v” if the stream contains protected or restricted MPEG-4 visual material).

8.12.3 IPMPInfoBox

(empty sub-clause)

8.12.4 IPMP Control Box

(empty sub-clause)

8.12.5 Scheme Type Box

8.12.5.1 Definition

Box Types: `'schm'`
Container: Protection Scheme Information Box (`'sinf'`), Restricted Scheme Information Box (`'rinf'`), or SRTP Process box (`'srpp'`)
Mandatory: No
Quantity: Zero or one in ‘sinf’, depending on the protection structure; Exactly one in ‘rinf’ and ‘srpp’

The Scheme Type Box (`'schm'`) identifies the protection or restriction scheme.

8.12.5.2 Syntax

```
aligned(8) class SchemeTypeBox extends FullBox('schm', 0, flags) {
   unsigned int(32)  scheme_type;    // 4CC identifying the scheme
   unsigned int(32)  scheme_version; // scheme version
   if (flags & 0x000001) {
      unsigned int(8)   scheme_uri[];       // browser uri
   }
}
```

8.12.5.3 Semantics

`scheme_type` is the code defining the protection or restriction scheme.
`scheme_version` is the version of the scheme (used to create the content)
`scheme_URI` allows for the option of directing the user to a web-page if they do not have the scheme installed on their system. It is an absolute URI formed as a null-terminated string in UTF-8 characters.

8.12.6 Scheme Information Box

8.12.6.1 Definition

Box Types: `'schi'`
Container: Protection Scheme Information Box (`'sinf'`), Restricted Scheme Information Box (`'rinf'`), or SRTP Process box (`'srpp'`)
Mandatory: No
Quantity: Zero or one

The Scheme Information Box is a container Box that is only interpreted by the scheme being used. Any information the encryption or restriction system needs is stored here. The content of this box is a series of boxes whose type and format are defined by the scheme declared in the Scheme Type Box.

8.12.6.2 Syntax

```
aligned(8) class SchemeInformationBox extends Box('schi') {
   Box   scheme_specific_data[];
}
```

8.13 File Delivery Format Support

8.13.1 Introduction

Files intended for transmission over ALC/LCT or FLUTE are stored as items in a top-level meta box (`'meta'`). The item location box (`'iloc'`) specifies the actual storage location of each item within the container file as well as the file size of each item. File name, content type (MIME type), etc., of each item are provided by version 1 of the item information box (`'iinf'`).

Pre-computed FEC reservoirs are stored as additional items in the meta box. If a source file is split into several source blocks, FEC reservoirs for each source block are stored as separate items. The relationship between FEC reservoirs and original source items is recorded in the partition entry box (`'paen'`) located in the FD item information box (`'fiin'`).

Pre-composed File reservoirs are stored as additional items in the container file. If a source file is split into several source blocks, each source block is stored as a separate item called a File reservoir. The relationship between File reservoirs and original source items is recorded in the partition entry box ('paen') located in the FD item information box ('fiin').

See subclause 9.2 for more details on the usage of the file delivery format.

8.13.2 FD Item Information Box

8.13.2.1 Definition

Box Type: 'fiin'
Container: Meta Box ('meta')
Mandatory: No
Quantity: Zero or one

The FD item information box is optional, although it is mandatory for files using FD hint tracks. It provides information on the partitioning of source files and how FD hint tracks are combined into FD sessions. Each partition entry provides details on a particular file partitioning, FEC encoding and associated File and FEC reservoirs. It is possible to provide multiple entries for one source file (identified by its item ID) if alternative FEC encoding schemes or partitionings are used in the file. All partition entries are implicitly numbered and the first entry has number 1.

8.13.2.2 Syntax

```
aligned(8) class PartitionEntry extends Box('paen') {
   FilePartitionBox  blocks_and_symbols;
   FECReservoirBox   FEC_symbol_locations; //optional
   FileReservoirBox  File_symbol_locations; //optional
}

aligned(8) class FDItemInformationBox
      extends FullBox('fiin', version = 0, 0) {
   unsigned int(16)  entry_count;
   PartitionEntry    partition_entries[ entry_count ];
   FDSessionGroupBox session_info;     //optional
   GroupIdToNameBox  group_id_to_name; //optional
}
```

8.13.2.3 Semantics

`entry_count` provides a count of the number of entries in the following array.

The semantics of the boxes are described where the boxes are documented.

8.13.3 File Partition Box

8.13.3.1 Definition

Box Type: 'fpar'
Container: Partition Entry ('paen')
Mandatory: Yes
Quantity: Exactly one

The File Partition box identifies the source file and provides a partitioning of that file into source blocks and symbols. Further information about the source file, e.g., filename, content location and group IDs, is contained in the Item Information box ('iinf'), where the Item Information entry corresponding to the item ID of the source file is of version 1 and includes a File Delivery Item Information Extension ('fdel').

8.13.3.2 Syntax

```
aligned(8) class FilePartitionBox
      extends FullBox('fpar', version = 0, 0) {
   unsigned int(16)  item_ID;
   unsigned int(16)  packet_payload_size;
   unsigned int(8)   reserved = 0;
   unsigned int(8)   FEC_encoding_ID;
   unsigned int(16)  FEC_instance_ID;
   unsigned int(16)  max_source_block_length;
   unsigned int(16)  encoding_symbol_length;
   unsigned int(16)  max_number_of_encoding_symbols;
   string            scheme_specific_info;
   unsigned int(16)  entry_count;
   for (i=1; i <= entry_count; i++) {
      unsigned int(16)  block_count;
      unsigned int(32)  block_size;
   }
}
```

8.13.3.3 Semantics

`item_ID` references the item in the item location box (`'iloc'`) that the file partitioning applies to.

`packet_payload_size` gives the target ALC/LCT or FLUTE packet payload size of the partitioning algorithm. Note that UDP packet payloads are larger, as they also contain ALC/LCT or FLUTE headers.

`FEC_encoding_ID` identifies the FEC encoding scheme and is subject to IANA registration (see RFC 5052). Note that i) value zero corresponds to the "Compact No-Code FEC scheme" also known as "Null-FEC" (RFC 3695); ii) value one corresponds to the “MBMS FEC” (3GPP TS 26.346); iii) for values in the range of 0 to 127, inclusive, the FEC scheme is Fully-Specified, whereas for values in the range of 128 to 255, inclusive, the FEC scheme is Under-Specified.

`FEC_instance_ID` provides a more specific identification of the FEC encoder being used for an Under-Specified FEC scheme. This value should be set to zero for Fully-Specified FEC schemes and shall be ignored when parsing a file with `FEC_encoding_ID` in the range of 0 to 127, inclusive. `FEC_instance_ID` is scoped by the `FEC_encoding_ID`. See RFC 5052 for further details.

`max_source_block_length` gives the maximum number of source symbols per source block.

`encoding_symbol_length` gives the size (in bytes) of one encoding symbol. All encoding symbols of one item have the same length except the last symbol which may be shorter.

`max_number_of_encoding_symbols` gives the maximum number of encoding symbols that can be generated for a source block for those FEC schemes in which the maximum number of encoding symbols is relevant, such as FEC encoding ID 129 defined in RFC 5052. For those FEC schemes in which the maximum number of encoding symbols is not relevant, the semantics of this field is unspecified.

`scheme_specific_info` is a base64-encoded null-terminated string of the scheme-specific object transfer information (FEC-OTI-Scheme-Specific-Info). The definition of the information depends on the FEC encoding ID.

`entry_count` gives the number of entries in the list of (`block_count`, `block_size`) pairs that provides a partitioning of the source file. Starting from the beginning of the file, each entry indicates how the next segment of the file is divided into source blocks and source symbols.

`block_count` indicates the number of consecutive source blocks of size `block_size`.

`block_size` indicates the size of a block (in bytes). A block_size that is not a multiple of the encoding_symbol_length symbol size indicates with Compact No-Code FEC that the last source symbols includes padding that is not stored in the item. With MBMS FEC (3GPP TS 26.346) the padding may extend across multiple symbols but the size of padding should never be more than encoding_symbol_length.

8.13.4 FEC Reservoir Box

8.13.4.1 Definition

Box Type: 'fecr'
Container: Partition Entry ('paen')
Mandatory: No
Quantity: Zero or One

The FEC reservoir box associates the source file identified in the file partition box ('fpar') with FEC reservoirs stored as additional items. It contains a list that starts with the first FEC reservoir associated with the first source block of the source file and continues sequentially through the source blocks of the source file.

8.13.4.2 Syntax

```
aligned(8) class FECReservoirBox
      extends FullBox('fecr', version = 0, 0) {
   unsigned int(16)  entry_count;
   for (i=1; i <= entry_count; i++) {
      unsigned int(16)  item_ID;
      unsigned int(32)  symbol_count;
   }
}
```

8.13.4.3 Semantics

`entry_count` gives the number of entries in the following list. An entry count here should match the total number of blocks in the corresponding file partition box.
`item_ID` indicates the location of the FEC reservoir associated with a source block.
`symbol_count` indicates the number of repair symbols contained in the FEC reservoir.

8.13.5 FD Session Group Box

8.13.5.1 Definition

Box Type: 'segr'
Container: FD Information Box ('fiin')
Mandatory: No
Quantity: Zero or One

The FD session group box is optional, although it is mandatory for files containing more than one FD hint track. It contains a list of sessions as well as all file groups and hint tracks that belong to each session. An FD session sends simultaneously over all FD hint tracks (channels) that are listed in the FD session group box for a particular FD session.

Only one session group should be processed at any time. The first listed hint track in a session group specifies the base channel. If the server has no preference between the session groups, the default choice should be the first session group. The group IDs of all file groups containing the files referenced by the hint tracks shall be included in the list of file groups. The file group IDs can in turn be translated into file group names (using the group ID to name box) that can be included by the server in FDTs.

8.13.5.2 Syntax

```
aligned(8) class FDSessionGroupBox extends Box('segr') {
   unsigned int(16)  num_session_groups;
   for(i=0; i < num_session_groups; i++) {
      unsigned int(8)   entry_count;
      for (j=0; j < entry_count; j++) {
         unsigned int(32)  group_ID;
      }
      unsigned int(16) num_channels_in_session_group;
      for(k=0; k < num_channels_in_session_group; k++) {
         unsigned int(32) hint_track_id;
      }
   }
}
```

8.13.5.3 Semantics

`num_session_groups` specifies the number of session groups.
`entry_count` gives the number of entries in the following list comprising all file groups that the session group complies with. The session group contains all files included in the listed file groups as specified by the item information entry of each source file. Note that the FDT for the session group should only contain those groups that are listed in this structure.
`group_ID` indicates a file group that the session group complies with.
`num_channels_in_session_groups` specifies the number of channels in the session group. The value of `num_channels_in_session_groups` shall be a positive integer.
`hint_track_ID` specifies the track ID of the FD hint track belonging to a particular session group. Note that one FD hint track corresponds to one LCT channel.

8.13.6 Group ID to Name Box

8.13.6.1 Definition

Box Type: `'gitn'`
Container: FD Information Box (`'fiin'`)
Mandatory: No
Quantity: Zero or One

The Group ID to Name box associates file group names to file group IDs used in the version 1 item information entries in the item information box (`'iinf'`).

8.13.6.2 Syntax

```
aligned(8) class GroupIdToNameBox
      extends FullBox('gitn', version = 0, 0) {
   unsigned int(16)  entry_count;
   for (i=1; i <= entry_count; i++) {
      unsigned int(32)  group_ID;
      string            group_name;
   }
}
```

8.13.6.3 Semantics

`entry_count` gives the number of entries in the following list.
`group_ID` indicates a file group.
`group_name` is a null-terminated string in UTF-8 characters containing a file group name.

8.13.7 File Reservoir Box

8.13.7.1 Definition

Box Type: 'fire'
Container: Partition Entry ('paen')
Mandatory: No
Quantity: Zero or One

The File reservoir box associates the source file identified in the file partition box ('fpar') with File reservoirs stored as additional items. It contains a list that starts with the first File reservoir associated with the first source block of the source file and continues sequentially through the source blocks of the source file.

8.13.7.2 Syntax

```
aligned(8) class FileReservoirBox
      extends FullBox('fire', version = 0, 0) {
   unsigned int(16)  entry_count;
   for (i=1; i <= entry_count; i++) {
      unsigned int(16)  item_ID;
      unsigned int(32)  symbol_count;
   }
}
```

8.13.7.3 Semantics

`entry_count` gives the number of entries in the following list. An entry count here should match the total number or blocks in the corresponding file partition box.
`item_ID` indicates the location of the File reservoir associated with a source block.
`symbol_count` indicates the number of source symbols contained in the File reservoir.

8.14 Sub tracks

8.14.1 Introduction

Sub tracks are used to assign parts of tracks to alternate and switch groups in the same way as (entire) tracks can be assigned to alternate and switch groups to indicate whether those tracks are alternatives to each other and whether it makes sense to switch between them during a session. Sub tracks are suitable for layered media, e.g., SVC and MVC, where media alternatives often are incommensurate with track structures. By defining alternate and switch groups at sub-track level it is possible to use existing rules for media selection and switching for such layered codecs. The over-all syntax is generic for all kinds of media and backward compatible with track-level definitions. Sub-track level alternate and switch groups use the same numbering as track level groups. The numberings are global over all tracks such that groups can be defined across track and sub-track boundaries.

In order to define sub tracks, media-specific definitions are required. Definitions for SVC and MVC are specified in the AVC file format (ISO/IEC 14496-15). Another way is to define sample groups and map them to sub tracks using the Sub Track Sample Group box defined here. The syntax can also be extended to include other media-specific definitions.

For each sub track that shall be defined a Sub Track box shall be included in the User Data box of the corresponding track. The Sub Track box contains objects that define and provide information about a sub track in the same track. The Track Selection box for this same track is already located here.

8.14.2 Backward compatibility

The default is to assign alternate and switch groups to 0 (zero) for (entire) tracks, which means that there is no information on alternate and/or switch groups for those (entire) tracks. However, file readers that are aware of sub-track definitions will be able to find sub-track information on alternate and switch groups even if the track indication is set to 0. This way it is possible to indicate that a file can be used by legacy readers by including the appropriate brand in the file type box. A file creator that requires a reader to be aware of sub-track information should not include legacy brands.

The same method of assigning sub track information can also be applied if all parts of a track except a sub track belong to the same alternate or switch group. Then the overall definitions can be made on track level as usual and specific assignments can be made at sub-track level. For sub tracks without specific assignments, track level assignments apply by default. As before, if a file creator requires a reader to be aware of sub-track information it should not include legacy brands (which would otherwise indicate that sub track information can be skipped).

8.14.3 Sub Track box

8.14.3.1 Definition

Box Type: 'strk'
Container: User Data box ('udta') of the corresponding Track box ('trak')
Mandatory: No
Quantity: Zero or more

This box contains objects that define and provide information about a sub track in the present track.

8.14.3.2 Syntax

```
aligned(8) class SubTrack extends Box('strk') {
}
```

8.14.4 Sub Track Information box

8.14.4.1 Definition

Box Type: 'stri'
Container: Sub Track box ('strk')
Mandatory: Yes
Quantity: One

8.14.4.2 Syntax

```
aligned(8) class SubTrackInformation
   extends FullBox('stri', version = 0, 0){
   template int(16)  switch_group = 0;
   template int(16)  alternate_group = 0;
   template unsigned int(32)  sub_track_ID = 0;
   unsigned int(32)  attribute_list[];  // to the end of the box
}
```

8.14.4.3 Semantics

`switch_group` is an integer that specifies a group or collection of tracks and/or sub tracks. If this field is 0 (default value), then there is no information on whether the sub track can be used for switching during playing or streaming. If this integer is not 0 it shall be the same for tracks and/or sub tracks that can be used for switching between each other. Tracks that belong to the same switch group shall belong to the same alternate group. A switch group may have only one member.

`alternate_group` is an integer that specifies a group or collection of tracks and/or sub tracks. If this field is 0 (default value), then there is no information on possible relations to other tracks/sub-tracks. If this field is not 0, it should be the same for tracks/sub-tracks that contain alternate data for one another and different for tracks/sub-tracks belonging to different such groups. Only one track/sub-track within an alternate group should be played or streamed at any one time.

`sub_track_ID` is an integer. A non-zero value uniquely identifies the sub track locally within the track. A zero value (default) means that sub track ID is not assigned.

`attribute_list` is a list, to the end of the box, of attributes. The attributes in this list should be used as descriptions of sub tracks or differentiating criteria for tracks and sub tracks in the same alternate or switch group.

The following attributes are descriptive:

Name	*Attribute*	*Description*
Temporal scalability	`'tesc'`	The sub-track can be temporally scaled.
Fine-grain SNR scalability	`'fgsc'`	The sub-track can be scaled in terms of quality.
Coarse-grain SNR scalability	`'cgsc'`	The sub-track can be scaled in terms of quality.
Spatial scalability	`'spsc'`	The sub-track can be spatially scaled.
Region-of-interest scalability	`'resc'`	The sub-track can be region-of-interest scaled.
View scalability	`'vwsc'`	The sub-track can be scaled in terms of number of views.

The following attributes are differentiating:

Name	*Attribute*	*Pointer*
Bitrate	`'bitr'`	Total size of the samples in the track divided by the duration in the track header box
Frame rate	`'frar'`	Number of samples in the track divided by duration in the track header box
Number of views	`'nvws'`	Number of views in the sub track

8.14.5 Sub Track Definition box

8.14.5.1 Definition

Box Type: `'strd'`
Container: Sub Track box (`'strk'`)
Mandatory: Yes
Quantity: One

This box contains objects that provide a definition of the sub track.

8.14.5.2 Syntax

```
aligned(8) class SubTrackDefinition extends Box('strd') {
}
```

8.14.6 Sub Track Sample Group box

8.14.6.1 Definition

Box Type: 'stsg'
Container: Sub Track Definition box ('strd')
Mandatory: No
Quantity: Zero or more

This box defines a sub track as one or more sample groups by referring to the corresponding sample group descriptions describing the samples of each group.

8.14.6.2 Syntax

```
aligned(8) class SubTrackSampleGroupBox
   extends FullBox('stsg', 0, 0){
   unsigned int(32) grouping_type;
   unsigned int(16) item_count;
   for(i = 0; i< item_count; i++)
      unsigned int(32)  group_description_index;
}
```

8.14.6.3 Semantics

`grouping_type` is an integer that identifies the sample grouping. The value shall be the same as in the corresponding `SampletoGroup` and `SampleGroupDescription` boxes.
`item_count` counts the number of sample groups listed in this box.
`group_description_index` is an integer that gives the index of the sample group entry which describes the samples in the group.

8.15 Post-decoder requirements on media

8.15.1 General

In order to handle situations where the file author requires certain actions on the player or renderer, this Subclause specifies a mechanism that enables players to simply inspect a file to find out such requirements for rendering a bitstream and stops legacy players from decoding and rendering files that require further processing. The mechanism applies to any type of video codec. In particular it applies to AVC and for this case specific signaling is defined in the AVC file format (ISO/IEC 14496-15) that allows a file author to list occurring SEI message IDs and distinguish between required and non-required actions for the rendering process.

The mechanism is similar to the content protection transformation where sample entries are hidden behind generic sample entries, 'encv', 'enca', etc., indicating encrypted or encapsulated media. The analogous mechanism for restricted video uses a transformation with the generic sample entry 'resv'. The method may be applied when the content should only be decoded by players that present it correctly.

8.15.2 Transformation

The method is applied as follows:

1) The four-character-code of the sample entry is replaced by a new sample entry code 'resv' meaning restricted video.

2) A Restricted Scheme Info box is added to the sample description, leaving all other boxes unmodified.

3) The original sample entry type is stored within an Original Format box contained in the Restricted Scheme Info box.

A `RestrictedSchemeInfoBox` is formatted exactly the same as a `ProtectionSchemeInfoBox`, except that is uses the identifier `'rinf'` instead of `'sinf' (see below)`.

The original sample entry type is contained in the Original Format box located in the Restricted Scheme Info box (in an identical way to the Protection Scheme Info box for encrypted media).

The exact nature of the restriction is defined in the `SchemeTypeBox`, and the data needed for that scheme is stored in the `SchemeInformationBox`, again, analogously to protection information.

Note that restriction and protection can be applied at the same time. The order of the transformations follows from the four-character code of the sample entry. For instance, if the sample entry type is `'resv'`, undoing the above transformation may result in a sample entry type `'encv'`, indicating that the media is protected.

Note that if the file author only wants to provide advisory information without stopping legacy players from playing the file, the Restricted Scheme Info box may be placed inside the sample entry without applying the four-character-code transformation. In this case it is not necessary to include an Original Format box.

8.15.3 Restricted Scheme Information box

8.15.3.1 Definition

Box Types: `'rinf'`
Container: Restricted Sample Entry or Sample Entry
Mandatory: Yes
Quantity: Exactly one

The Restricted Scheme Information Box contains all the information required both to understand the restriction scheme applied and its parameters. It also documents the original (un-transformed) sample entry type of the media. The Restricted Scheme Information Box is a container Box. It is mandatory in a sample entry that uses a code indicating a restricted stream, i.e., `'resv'`.

When used in a restricted sample entry, this box must contain the original format box to document the original sample entry type and a Scheme type box. A Scheme Information box may be required depending on the restriction scheme.

8.15.3.2 Syntax

```
aligned(8) class RestrictedSchemeInfoBox(fmt) extends Box('rinf') {
   OriginalFormatBox(fmt)  original_format;
   SchemeTypeBox           scheme_type_box;
   SchemeInformationBox    info;        // optional
}
```

8.15.4 Scheme for stereoscopic video arrangements

8.15.4.1 General

When stereo-coded video frames are decoded, the decoded frames either contain a representation of two spatially packed constituent frames that form a stereo pair (frame packing) or only one view of a stereo pair (left and right views in different tracks). Restrictions due to stereo-coded video are contained in the Stereo Video box.

The SchemeType `'stvi'` (stereoscopic video) is used.

8.15.4.2 Stereo video box

8.15.4.2.1 Definition

Box Type: `stvi'
Container: Scheme Information box (`schi')
Mandatory: Yes (when the SchemeType is `stvi')
Quantity: One

The Stereo Video box is used to indicate that decoded frames either contain a representation of two spatially packed constituent frames that form a stereo pair or contain one of two views of a stereo pair. The Stereo Video box shall be present when the SchemeType is `stvi'.

8.15.4.2.2 Syntax

```
aligned(8) class StereoVideoBox extends extends FullBox('stvi', version = 0, 0)
{
   template unsigned int(30) reserved = 0;
   unsigned int(2)   single_view_allowed;
   unsigned int(32)  stereo_scheme;
   unsigned int(32)  length;
   unsigned int(8)[length] stereo_indication_type;
   Box[] any_box; // optional
}
```

8.15.4.2.3 Semantics

`single_view_allowed` is an integer. A zero value indicates that the content may only be displayed on stereoscopic displays. When (`single_view_allowed` & 1) is equal to 1, it is allowed to display the right view on a monoscopic single-view display. When (`single_view_allowed` & 2) is equal to 2, it is allowed to display the left view on a monoscopic single-view display.

`stereo_scheme` is an integer that indicates the stereo arrangement scheme used and the stereo indication type according to the used scheme. The following values for `stereo_scheme` are specified:

1: the frame packing scheme as specified by the Frame packing arrangement Supplemental Enhancement Information message of ISO/IEC 14496-10 [ISO/IEC 14496-10]
2: the arrangement type scheme as specified in Annex L of ISO/IEC 13818-2 [ISO/IEC 13818-2:2000/Amd.4]
3: the stereo scheme as specified in ISO/IEC 23000-11 for both frame/service compatible and 2D/3D mixed services.

Other values of `stereo_scheme` are reserved.

`length` indicates the number of bytes for the `stereo_indication_type` field.

`stereo_indication_type` indicates the stereo arrangement type according to the used stereo indication scheme. The syntax and semantics of `stereo_indication_type` depend on the value of `stereo_scheme`. The syntax and semantics for `stereo_indication_type` for the following values of `stereo_scheme` are specified as follows:

`stereo_scheme` equal to 1: The value of `length` shall be 4 and `stereo_indication_type` shall be `unsigned int(32)` which contains the frame_packing_arrangement_type value from Table D-8 of ISO/IEC 14496-10 [ISO/IEC 14496-10] ('Definition of frame_packing_arrangement_type').

`stereo_scheme` equal to 2: The value of `length` shall be 4 and `stereo_indication_type` shall be `unsigned int(32)` which contains the type value from Table L-1 of ISO/IEC 13818-2 [ISO/IEC 13818-2:2000/Amd.4] ('Definition of arrangement_type').

`stereo_scheme` equal to 3: The value of `length` shall be 2 and `stereo_indication_type` shall contain two syntax elements of `unsigned int(8)`. The first syntax element shall contain the stereoscopic composition type from Table 4 of ISO/IEC 23000-11:2009. The least significant bit of the second syntax element shall contain the value of `is_left_first` as specified in 8.4.3 of ISO/IEC 23000-11:2009, while the other bits are reserved and shall be set to 0.

The following applies when the Stereo Video box is used:

— In the Track Header box

  — `width` and `height` specify the visual presentation size of a single view after unpacking.

— In the Sample Description box

  — `frame_count` shall be 1, because the decoder physically outputs a single frame. In other words, the constituent frames included within a frame-packed picture are not documented by `frame_count`.

  — `width` and `height` document the pixel counts of a frame-packed picture (and not the pixel counts of a single view within a frame-packed picture).

  — the Pixel Aspect Ratio box documents the pixel aspect ratio of each view when the view is displayed on a monoscopic single-view display. For example, in many spatial frame packing arrangements, the Pixel Aspect Ratio box therefore indicates 2:1 or 1:2 pixel aspect ratio, as the spatial resolution of one view of frame-packed video is typically halved along one coordinate axis compared to that of the single-view video of the same format.

8.16 Segments

8.16.1 Introduction

Media presentations may be divided into segments for delivery, for example, it is possible (e.g. in HTTP streaming) to form files that contain a segment – or concatenated segments – which would not necessarily form ISO base media file format compliant files (e.g. they do not contain a movie box).

This Subclause defines specific boxes that may be used in such segments.

8.16.2 Segment Type Box

Box Type: `styp'
Container: File
Mandatory: No
Quantity: Zero or more

If segments are stored in separate files (e.g. on a standard HTTP server) it is recommended that these 'segment files' contain a segment-type box, which must be first if present, to enable identification of those files, and declaration of the specifications with which they are compliant.

A segment type has the same format as an `'ftyp'` box [4.3], except that it takes the box type `'styp'`. The brands within it may include the same brands that were included in the `'ftyp'` box that preceded the `'moov'` box, and may also include additional brands to indicate the compatibility of this segment with various specification(s).

Valid segment type boxes shall be the first box in a segment. Segment type boxes may be removed if segments are concatenated (e.g. to form a full file), but this is not required. Segment type boxes that are not first in their files may be ignored.

8.16.3 Segment Index Box

8.16.3.1 Definition

Box Type: `sidx'
Container: File
Mandatory: No
Quantity: Zero or more

The Segment Index box ('sidx') provides a compact index of one media stream within the media segment to which it applies. It is designed so that it can be used not only with media formats based on this specification (i.e. segments containing sample tables or movie fragments), but also other media formats (for example, MPEG-2 Transport Streams [ISO/IEC 13818-1]). For this reason, the formal description of the box given here is deliberately generic, and then at the end of this Subclause the specific definitions for segments using movie fragments are given.

Each Segment Index box documents how a (sub)segment is divided into one or more subsegments (which may themselves be further subdivided using Segment Index boxes).

A subsegment is defined as a time interval of the containing (sub)segment, and corresponds to a single range of bytes of the containing (sub)segment. The durations of all the subsegments sum to the duration of the containing (sub)segment.

Each entry in the Segment Index box contains a reference type that indicates whether the reference points directly to the media bytes of a referenced leaf subsegment, or to a Segment Index box that describes how the referenced subsegment is further subdivided; as a result, the segment may be indexed in a 'hierarchical' or 'daisy-chain' or other form by documenting time and byte offset information for other Segment Index boxes applying to portions of the same (sub)segment.

Each Segment Index box provides information about a single media stream of the Segment, referred to as the reference stream. If provided, the first Segment Index box in a segment, for a given media stream, shall document the entirety of that media stream in the segment, and shall precede any other Segment Index box in the segment for the same media stream.

If a segment index is present for at least one media stream but not all media streams in the segment, then normally a media stream in which not every access unit is independently coded, such as video, is selected to be indexed. For any media stream for which no segment index is present, referred to as non-indexed stream, the media stream associated with the first Segment Index box in the segment serves as a reference stream in a sense that it also describes the subsegments for any non-indexed media stream.

NOTE 1 Further restrictions may be specified in derived specifications.

Segment Index boxes may be inline in the same file as the indexed media or, in some cases, in a separate file containing only indexing information.

A Segment Index box contains a sequence of references to subsegments of the (sub)segment documented by the box. The referenced subsegments are contiguous in presentation time. Similarly, the bytes referred to by a Segment Index box are always contiguous in both the media file, and the separate index segment, or in the single file if indexes are placed within the media file. The referenced size gives the count of the number of bytes in the material referenced.

NOTE 2 A media segment may be indexed by more than one "top-level" Segment Index box that are independent of each other, each of which indexes one media stream within the media segment. In segments containing multiple media streams the referenced bytes may contain media from multiple streams, even though the Segment Index box provides timing information for only one media stream.

In the file containing the Segment Index box, the anchor point for a Segment Index box is the first byte after that box. If there are two files, the anchor point in the media file is the beginning of the top-level segment (i.e. the beginning of the segment file if each segment is stored in a separate file). The material in the file

containing media (which may also be the file that contains the segment index boxes) starts at the indicated offset from the anchor point. If there are two files, the material in the index file starts at the anchor point, i.e. immediately following the Segment Index box.

Within the two constraints (a) that, in time, the subsegments are contiguous, that is, each entry in the loop is consecutive from the immediately preceding one and (b) within a given file (integrated file, media file, or index side file) the referenced bytes are contiguous, there are a number of possibilities, including:

1) a reference to a segment index box may include, in its byte count, immediately following Segment Index boxes that document subsegments;

2) in an integrated file, using the `first_offset` field, it is possible to separate Segment Index boxes from the media that they refer to;

3) in an integrated file, it is possible to locate Segment Index boxes for subsegments close to the media they index;

4) when a separate file containing Segment Indexes is used, it is possible for the loop entries to be of 'mixed type', some to Segment Index boxes in the index segment, some to media subsegments in the media file.

NOTE 3 Profiles may be used to restrict the placement of segment indexes, or the overall complexity of the indexing.

The Segment Index box documents the presence of Stream Access Points (SAPs), as specified in Annex I, in the referenced subsegments. The annex specifies characteristics of SAPs, such as I_{SAU}, I_{SAP} and T_{SAP}, as well as SAP types, which are all used in the semantics below. A subsegment starts with a SAP when the subsegment contains a SAP, and for the first SAP, I_{SAU} is the index of the first access unit that follows I_{SAP}, and I_{SAP} is contained in the subsegment.

For segments based on this specification (i.e. based on movie sample tables or movie fragments):

- an access unit is a sample;
- a subsegment is a self-contained set of one or more consecutive movie fragments; a self-contained set contains one or more Movie Fragment boxes with the corresponding Media Data box(es), and a Media Data Box containing data referenced by a Movie Fragment Box must follow that Movie Fragment box and precede the next Movie Fragment box containing information about the same track;
- Segment Index boxes shall be placed before subsegment material they document, that is, before any Movie Fragment ('moof') box of the documented material of the subsegment;
- streams are tracks in the file format, and stream IDs are track IDs;
- a subsegment contains a stream access point if a track fragment within the subsegment for the track with `track_ID` equal to `reference_ID` contains a stream access point;
- initialisation data for SAPs consists of the movie box;
- presentation times are in the movie timeline, that is they are composition times after the application of any edit list for the track;
- the I_{SAP} is a position exactly pointing to the start of a top-level box, such as a movie fragment box 'moof';
- a SAP of type 1 or type 2 is indicated as a sync sample, or by `sample_is_non_sync_sample` equal to 0 in the movie fragment;
- a SAP of type 3 is marked as a member of a sample group of type `'rap '`;
- a SAP of type 4 is marked as a member of a sample group of type `'roll'` where the value of the `roll_distance` field is greater than 0.

NOTE 4 For SAPs of type 5 and 6, no specific signalling in the ISO base media file format is supported.

8.16.3.2 Syntax

```
aligned(8) class SegmentIndexBox extends FullBox('sidx', version, 0) {
   unsigned int(32) reference_ID;
   unsigned int(32) timescale;
   if (version==0) {
         unsigned int(32) earliest_presentation_time;
         unsigned int(32) first_offset;
      }
      else {
         unsigned int(64) earliest_presentation_time;
         unsigned int(64) first_offset;
      }
   unsigned int(16) reserved = 0;
   unsigned int(16) reference_count;
   for(i=1; i <= reference_count; i++)
   {
      bit (1)             reference_type;
      unsigned int(31)    referenced_size;
      unsigned int(32)    subsegment_duration;
      bit(1)              starts_with_SAP;
      unsigned int(3)     SAP_type;
      unsigned int(28)    SAP_delta_time;
   }
}
```

8.16.3.3 Semantics

`reference_ID` provides the stream ID for the reference stream; if this Segment Index box is referenced from a "parent" Segment Index box, the value of `reference_ID` shall be the same as the value of `reference_ID` of the "parent" Segment Index box;

`timescale` provides the timescale, in ticks per second, for the time and duration fields within this box; it is recommended that this match the timescale of the reference stream or track; for files based on this specification, that is the timescale field of the Media Header Box of the track;

`earliest_presentation_time` is the earliest presentation time of any access unit in the reference stream in the first subsegment, in the timescale indicated in the timescale field;

`first_offset` is the distance in bytes, in the file containing media, from the anchor point, to the first byte of the indexed material;

`reference_count` provides the number of referenced items;

`reference_type`: when set to 1 indicates that the reference is to a segment index (`'sidx'`) box; otherwise the reference is to media content (e.g., in the case of files based on this specification, to a movie fragment box); if a separate index segment is used, then entries with reference type 1 are in the index segment, and entries with reference type 0 are in the media file;

`referenced_size`: the distance in bytes from the first byte of the referenced item to the first byte of the next referenced item, or in the case of the last entry, the end of the referenced material;

`subsegment_duration`: when the reference is to Segment Index box, this field carries the sum of the `subsegment_duration` fields in that box; when the reference is to a subsegment, this field carries the difference between the earliest presentation time of any access unit of the reference stream in the next subsegment (or the first subsegment of the next segment, if this is the last subsegment of the segment, or the end presentation time of the reference stream if this is the last subsegment of the stream) and the earliest presentation time of any access unit of the reference stream in the referenced subsegment; the duration is in the same units as `earliest_presentation_time`;

`starts_with_SAP` indicates whether the referenced subsegments start with a SAP. For the detailed semantics of this field in combination with other fields, see the table below.

`SAP_type` indicates a SAP type as specified in Annex I, or the value 0. Other type values are reserved. For the detailed semantics of this field in combination with other fields, see the table below.

`SAP_delta_time`: indicates T_{SAP} of the first SAP, in decoding order, in the referenced subsegment for the reference stream. If the referenced subsegments do not contain a SAP, `SAP_delta_time` is reserved with the value 0; otherwise `SAP_delta_time` is the difference between the earliest presentation time of the subsegment, and the T_{SAP} (note that this difference may be zero, in the case that the subsegment starts with a SAP).

Table 4 — Semantics of SAP and reference type combinations

`starts_with_SAP`	`SAP_type`	`reference_type`	Meaning
0	0	0 or 1	No information of SAPs is provided.
0	1 to 6, inclusive	0 (media)	The subsegment contains (but may not start with) a SAP of the given `SAP_type` and the first SAP of the given `SAP_type` corresponds to `SAP_delta_time`.
0	1 to 6, inclusive	1 (index)	All the referenced subsegments contain a SAP of at most the given `SAP_type` and none of these SAPs is of an unknown type.
1	0	0 (media)	The subsegment starts with a SAP of an unknown type.
1	0	1 (index)	All the referenced subsegments start with a SAP which may be of an unknown type
1	1 to 6, inclusive	0 (media)	The referenced subsegment starts with a SAP of the given `SAP_type`.
1	1 to 6, inclusive	1 (index)	All the referenced subsegments start with a SAP of at most the given `SAP_type` and none of these SAPs is of an unknown type.

8.16.4 Subsegment Index Box

8.16.4.1 Definition

Box Type: `'ssix'`
Container: File
Mandatory: No
Quantity: Zero or more

The Subsegment Index box ('ssix') provides a mapping from levels (as specified by the Level Assignment box) to byte ranges of the indexed subsegment. In other words, this box provides a compact index for how the data in a subsegment is ordered according to levels into partial subsegments. It enables a client to easily access data for partial subsegments by downloading ranges of data in the subsegment.

Each byte in the subsegment shall be assigned to a level. If the range is not associated with any information in the level assignment, then any level that is not included in the level assignment may be used.

There shall be 0 or 1 Subsegment Index boxes per each Segment Index box that indexes only leaf subsegments, i.e. that only indexes subsegments but no segment indexes. A Subsegment Index box, if any, shall be the next box after the associated Segment Index box. A Subsegment Index box documents the subsegment that is indicated in the immediately preceding Segment Index box.

In general, the media data constructed from the byte ranges is incomplete, i.e. it does not conform to the media format of the entire subsegment.

For leaf subsegments based on this specification (i.e. based on movie sample tables and movie fragments):

- Each level shall be assigned to exactly one partial subsegment, i.e. byte ranges for one level shall be contiguous.

- Levels of partial subsegments shall be assigned by increasing numbers within a subsegment, i.e., samples of a partial subsegment may depend on any samples of preceding partial subsegments in the same subsegment, but not the other way around. For example, each partial subsegment contains samples having an identical temporal level and partial subsegments appear in increasing temporal level order within the subsegment.

- When a partial subsegment is accessed in this way, for any `assignment_type` other than 3, the final Media Data box may be incomplete, that is, less data is accessed than the length indication of the Media Data Box indicates is present. The length of the Media Data box may need adjusting, or padding used. The `padding_flag` in the Level Assignment Box indicates whether this missing data can be replaced by zeros. If not, the sample data for samples assigned to levels that are not accessed is not present, and care should be taken not to attempt to process such samples.

NOTE `assignment_type` equal to 0 (specified in the subsegment index box 'leva') can be used, for example, together with the temporal level sample grouping ('tele') when frames of a video bitstream are temporally ordered within subsegments; `assignment_type` equal to 2 can be used, for example, when each view of a multiview video bitstream is contained in a separate track and the track fragments for all the views are contained in a single movie fragment. `assignment_type` equal to 3 may be used, for example, when audio and video movie fragments (including the respective Media Data boxes) are interleaved. The first level can be specified to contain the audio movie fragments (including the respective Media Data boxes), whereas the second level can be specified to contain both audio and video movie fragments (including all Media Data boxes).

8.16.4.2 Syntax

```
aligned(8) class SubsegmentIndexBox extends FullBox('ssix', 0, 0) {
   unsigned int(32)  subsegment_count;
   for( i=1; i <= subsegment_count; i++)
   {
      unsigned int(32)  ranges_count;
      for ( j=1; j <= range_count; j++) {
         unsigned int(8) level;
         unsigned int(24) range_size;
      }
   }
}
```

8.16.4.3 Semantics

`subsegment_count` is a positive integer specifying the number of subsegments for which partial subsegment information is specified in this box. `subsegment_count` shall be equal to `reference_count` (i.e., the number of movie fragment references) in the immediately preceding Segment Index box.

`range_count` specifies the number of partial subsegment levels into which the media data is grouped. This value shall be greater than or equal to 2.

`range_size` indicates the size of the partial subsegment.

`level` specifies the level to which this partial subsegment is assigned.

8.16.5 Producer Reference Time Box

8.16.5.1 Definition

Box Type: `prft'
Container: File
Mandatory: No
Quantity: Zero or more

The producer reference time box supplies relative wall-clock times at which movie fragments, or files containing movie fragments (such as segments) were produced. When these files are both produced and consumed in real time, this can provide clients with information to enable consumption and production to proceed at equivalent rates, thus avoiding possible buffer overflow or underflow.

This box is related to the next movie fragment box that follows it in bitstream order. It must follow any segment type or segment index box (if any) in the segment, and occur before the following movie fragment box (to which it refers). If a segment file contains any producer reference time boxes, then the first of them shall occur before the first movie fragment box in that segment.

The box contains a time value measured on a clock which increments at the same rate as a UTC-synchronized NTP [RFC 5905] clock, using NTP format. This is associated with a media time for one of the tracks in the movie fragment. That media time should be in the range of times in that track in the associated movie fragment.

Producer reference times should be associated with at most one track.

8.16.5.2 Syntax

```
aligned(8) class ProducerReferenceTimeBox extends FullBox('prft', version, 0) {
   unsigned int(32) reference_track_ID;
   unsigned int(64) ntp_timestamp;
   if (version==0) {
      unsigned int(32)  media_time;
   } else {
      unsigned int(64)  media_time;
   }
}
```

8.16.5.3 Semantics

`reference_track_ID` provides the `track_ID` for the reference track.
`ntp_timestamp` indicates a UTC time in NTP format corresponding to `decoding_time`.
`media_time` corresponds to the same time as `ntp_timestamp`, but in the time units used for the reference track, and is measured on this media clock as the media is produced.

NOTE In most cases this timestamp will not be equal to the timestamp of the first sample of the adjacent segment of the reference track, but it is recommended it be in the range of the segment containing this producer reference time box.

9 Hint Track Formats

9.1 RTP and SRTP Hint Track Format

9.1.1 Introduction

RTP is the real-time transport protocol defined by the IETF (RFC 3550 and 3551) and is currently defined to be able to carry a limited set of media types (principally audio and video) and codings. The packing of MPEG-4 elementary streams into RTP is under discussion in both bodies. However, it is clear that the way the media is packetized does not differ in kind from the existing techniques used for other codecs in RTP, and supported by this scheme.

In standard RTP, each media stream is sent as a separate RTP stream; multiplexing is achieved by using IP's port-level multiplexing, not by interleaving the data from multiple streams into a single RTP session. However, if MPEG is used, it may be necessary to multiplex several media tracks into one RTP track (eg. when using MPEG-2 transport in RTP, or FlexMux). Each hint track is therefore tied to a set of media tracks by track references. The hint tracks extract data from their media tracks by indexing through this table. Hint track references to media tracks have the reference type `'hint'`.

This design decides the packet size at the time the server hint track is created; therefore, in the declarations for the hint track, we indicate the chosen packet size. This is in the sample-description. Note that it is valid for there to be several RTP hint tracks for each media track, with different packet size choices. Similarly the time-scale for the RTP clock is provided. The timescale of the server hint track is usually chosen to match the timescale of the media tracks, or a suitable value is picked for the server. In some cases, the RTP timescale is different (e.g. 90 kHz for some MPEG payloads), and this permits that variation. Session description (SAP/SDP) information is stored in user-data boxes in the track.

RTP hint tracks do not use the composition time offset table (`'ctts'`). Instead, the hinting process for server hint tracks establishes the correct transmission order and time-stamps, perhaps using the transmission time offset to set transmission times.

Hinted content may require the use of SRTP for streaming by using the hint track format for SRTP, defined here. SRTP hint tracks are formatted identically to RTP hint tracks, except that:

1) the sample entry name is changed from `'rtp '` to `'srtp'` to indicate to the server that SRTP is required;

2) an extra box is added to the sample entry which can be used to instruct the server in the nature of the on-the-fly encryption and integrity protection that must be applied.

9.1.2 Sample Description Format

RTP server hint tracks are hint tracks (media handler `'hint'`), with an entry-format in the sample description of `'rtp '`:

```
class RtpHintSampleEntry() extends SampleEntry ('rtp ') {
   uint(16)    hinttrackversion = 1;
   uint(16)    highestcompatibleversion = 1;
   uint(32)    maxpacketsize;
   box         additionaldata[];
}
```

The `hinttrackversion` is currently 1; the highest compatible version field specifies the oldest version with which this track is backward-compatible.

The `maxpacketsize` indicates the size of the largest packet that this track will generate.

The additional data is a set of boxes, from the following.

```
class timescaleentry() extends Box('tims') {
   uint(32) timescale;
}

class timeoffset() extends Box('tsro') {
   int(32)     offset;
}

class sequenceoffset extends Box('snro') {
   int(32)     offset;
}
```

The timescale entry is required. The other two are optional. The offsets over-ride the default server behaviour, which is to choose a random offset. A value of 0, therefore, will cause the server to apply no offset to the timestamp or sequence number respectively.

An SRTP Hint Sample entry is used when it is required that SRTP processing is required.

```
class SrtpHintSampleEntry() extends SampleEntry ('srtp'){
   uint(16)    hinttrackversion = 1;
   uint(16)    highestcompatibleversion = 1;
   uint(32)    maxpacketsize;
   box         additionaldata[];
}
```

Fields and boxes are defined as for the `RtpHintSampleEntry` (`'rtp '`) of the ISO Base Media File Format. However, an SRTP Process Box shall be included in an `SrtpHintSampleEntry` as one of the `additionaldata` boxes.

9.1.2.1 SRTP Process box `'srpp'`:

Box Type: `'srpp'`
Container: SrtpHintSampleEntry
Mandatory: Yes
Quantity: Exactly one

The SRTP Process Box may instruct the server as to which SRTP algorithms should be applied.

```
aligned(8) class SRTPProcessBox extends FullBox('srpp', version, 0) {
   unsigned int(32)  encryption_algorithm_rtp;
   unsigned int(32)  encryption_algorithm_rtcp;
   unsigned int(32)  integrity_algorithm_rtp;
   unsigned int(32)  integrity_algorithm_rtcp;
   SchemeTypeBox     scheme_type_box;
   SchemeInformationBox    info;
}
```

The Scheme Type Box and Scheme Information Box have the syntax defined above for protected media tracks. They serve to provide the parameters required for applying SRTP. The Scheme Type Box is used to indicate the necessary key-management and security policy for the stream in extension to the defined algorithmic pointers provided by the SRTPProcessBox. The key-management functionality is also used to establish all the necessary SRTP parameters as listed in section 8.2 of the SRTP specification. The exact definition of protection schemes is out of the scope of the file format.

The algorithms for encryption and integrity protection are defined by SRTP. The following format identifiers are defined here. An entry of four spaces (`$20$20$20$20`) may be used to indicate that the choice of algorithm for either encryption or integrity protection is decided by a process outside the file format.

Format	Algorithm
`$20$20$20$20`	The choice of algorithm for either encryption or integrity protection is decided by a process outside the file format
`ACM1`	Encryption using AES in Counter Mode with 128-bit key, as defined in Section 4.1.1 of the SRTP specification.
`AF81`	Encryption using AES in F8-mode with 128-bit key, as defined in Section 4.1.2 of the SRTP specification.
`ENUL`	Encryption using the NULL-algorithm as defined in Section 4.1.3 of the SRTP specification
`SHM2`	Integrity protection using HMAC-SHA-1 with 160-bit key, as defined in Section 4.2.1 of the SRTP specification.
`ANUL`	Integrity protection not applied to RTP (but still applied to RTCP). Note: this is valid only for `integrity_algorithm_rtp`

9.1.3 Sample Format

Each sample in a server hint track will generate one or more RTP packets, whose RTP timestamp is the same as the hint sample time. Therefore, all the packets made by one sample have the same timestamp. However, provision is made to ask the server to 'warp' the actual transmission times, for data-rate smoothing, for example.

Each sample contains two areas: the instructions to compose the packets, and any extra data needed when sending those packets (e.g. an encrypted version of the media data). Note that the size of the sample is known from the sample size table.

```
aligned(8) class RTPsample {
   unsigned int(16)  packetcount;
   unsigned int(16)  reserved;
   RTPpacket   packets[packetcount];
   byte     extradata[];
}
```

9.1.3.1 Packet Entry format

Each packet in the packet entry table has the following structure:

```
aligned(8) class RTPpacket {
   int(32)  relative_time;
   // the next fields form initialization for the RTP
   // header (16 bits), and the bit positions correspond
   bit(2)   RTP_version;
   bit(1)   P_bit;
   bit(1)   X_bit;
   bit(4)   CSRC_count;
   bit(1)   M_bit;
   bit(7)   payload_type;
   unsigned int(16)  RTPsequenceseed;
   unsigned int(13)  reserved = 0;
   unsigned int(1)   extra_flag;
   unsigned int(1)   bframe_flag;
   unsigned int(1)   repeat_flag;
   unsigned int(16)  entrycount;
   if (extra_flag) {
      uint(32) extra_information_length;
      box   extra_data_tlv[];
   }
   dataentry   constructors[entrycount];
}
```

The semantics of the fields for RTP server hint tracks is specified below. RTP reception hint tracks use the same packet structure. The semantics of the fields when the packet structure is used in an RTP reception hint track is specified in subclause 9.4.1.4.

In server hint tracks, the `relative_time` field 'warps' the actual transmission time away from the sample time. This allows traffic smoothing.

The following 2 bytes exactly overlay the RTP header; they assist the server in making the RTP header (the server fills in the remaining fields). Within these 2 bytes, the fields `RTP_version` and `CSRC_count` are reserved in server (transmission) hint tracks and the server fills in these fields.

The sequence seed is the basis for the RTP sequence number. If a hint track causes multiple copies of the same RTP packet to be sent, then the seed value would be the same for them all. The server normally adds a random offset to this value (but see above, under `'sequenceoffset'`).

`extra_flag` equal to 1 indicates that there is extra information before the constructors, in the form of type-length-value sets.

`extra_information_length` indicates the length in bytes of all extra information before the constructors, which includes the four bytes of the `extra information_length` field. The subsequent boxes before the constructors, referred to as the TLV boxes, are aligned on 32-bit boundaries. The box size of any TLV box indicates the actual bytes used, not the length required for padding to 32-bit boundaries. The value of `extra_information_length` includes the required padding for 32-bit boundaries.

`The rtpoffsetTLV ('rtpo')` gives a 32-bit signed integer offset to the actual RTP time-stamp to place in the packet. This enables packets to be placed in the hint track in decoding order, but have their presentation time-stamp in the transmitted packet be in a different order. This is necessary for some MPEG payloads.

The `bframe_flag` indicates a disposable 'b-frame'. The `repeat_flag` indicates a 'repeat packet', one that is sent as a duplicate of a previous packet. Servers may wish to optimize handling of these packets.

9.1.3.2 Constructor format

There are various forms of the constructor. Each constructor is 16 bytes, to make iteration easier. The first byte is a union discriminator:

```
aligned(8) class RTPconstructor(type) {
   unsigned int(8)   constructor_type = type;
}

aligned(8) class RTPnoopconstructor
   extends RTPconstructor(0)
{
   uint(8)  pad[15];
}

aligned(8) class RTPimmediateconstructor
   extends RTPconstructor(1)
{
   unsigned int(8)   count;
   unsigned int(8)   data[count];
   unsigned int(8)   pad[14 - count];
}

aligned(8) class RTPsampleconstructor
   extends RTPconstructor(2)
{
   signed int(8)  trackrefindex;
   unsigned int(16)  length;
   unsigned int(32)  samplenumber;
   unsigned int(32)  sampleoffset;
   unsigned int(16)  bytesperblock = 1;
   unsigned int(16)  samplesperblock = 1;
}

aligned(8) class RTPsampledescriptionconstructor
   extends RTPconstructor(3)
{
   signed int(8)  trackrefindex;
   unsigned int(16)  length;
   unsigned int(32)  sampledescriptionindex;
   unsigned int(32)  sampledescriptionoffset;
   unsigned int(32)  reserved;
}
```

The immediate mode permits the insertion of payload-specific headers (e.g. the RTP H.261 header). For hint tracks where the media is sent 'in the clear', the `sample` entry then specifies the bytes to copy from the media track, by giving the sample number, data offset, and length to copy. The track reference may index into the table of track references (a strictly positive value), name the hint track itself (-1), or the only associated media track (0). (The value zero is therefore equivalent to the value 1.)

The `bytesperblock` and `samplesperblock` concern compressed audio, using a scheme prior to MP4, in which the audio framing was not evident in the file. These fields have the fixed values of 1 for MP4 files.

The `sampledescription` mode allows sending of sample descriptions (which would contain elementary stream descriptors), by reference, as part of an RTP packet. The index is the index of a `SampleEntry` in a Sample Description Box, and the offset is relative to the beginning of that `SampleEntry`.

For complex cases (e.g. encryption or forward error correction), the transformed data would be placed into the hint samples, in the `extradata` field, and then sample mode referencing the hint track itself would be used.

Notice that there is no requirement that successive packets transmit successive bytes from the media stream. For example, to conform with RTP-standard packing of H.261, it is sometimes required that a byte be sent at the end of one packet and also at the beginning of the next (when a macroblock boundary falls within a byte).

9.1.4 SDP Information

Streaming servers using RTSP and SDP usually use SDP as the description format; and there are necessary relationships between the SDP information, and the RTP streams, such as the mapping of payload IDs to MIME names. Provision is therefore made for the hinter to leave fragments of SDP information in the file, to assist the server in forming a full SDP description. Note that there are required SDP entries which the server should also generate. The information here is only partial.

SDP information is formatted as a set of boxes within user-data boxes, at both the movie and the track level. The text in the movie-level SDP box should be placed before any media-specific lines (before the first 'm=' in the SDP file).

9.1.4.1 Movie SDP information

At the movie level, within the user-data (`'udta'`) box, a hint information container box may occur:

```
aligned(8) class moviehintinformation extends box('hnti') {
}

aligned(8) class rtpmoviehintinformation extends box('rtp ') {
   uint(32) descriptionformat = 'sdp ';
   char  sdptext[];
}
```

The hint information box may contain information for multiple protocols; only RTP is defined here. The RTP box may contain information for various description formats; only SDP is defined here. The `sdptext` is correctly formatted as a series of lines, each terminated by <crlf>, as required by SDP.

9.1.4.2 Track SDP Information

At the track level, the structure is similar; however, we already know that this track is an RTP hint track, from the sample description. Therefore the child box merely specifies the description format.

```
aligned(8) class trackhintinformation extends box('hnti') {
}

aligned(8) class rtptracksdphintinformation extends box('sdp ') {
   char  sdptext[];
}
```

The `sdptext` is correctly formatted as a series of lines, each terminated by <crlf>, as required by SDP.

9.1.5 Statistical Information

In addition to the statistics in the hint media header, the hinter may place extra data in a hint statistics box, in the track user-data box. This is a container box with a variety of sub-boxes that it may contain.

```
aligned(8) class hintstatisticsbox extends box('hinf') {
}

aligned(8) class hintBytesSent extends box('trpy') {
   uint(64) bytessent; }    // total bytes sent, including 12-byte RTP headers
aligned(8) class hintPacketsSent extends box('nump') {
   uint(64) packetssent; } // total packets sent
aligned(8) class hintBytesSent extends box('tpyl') {
   uint(64) bytessent; }    // total bytes sent, not including RTP headers

aligned(8) class hintBytesSent extends box('totl') {
   uint(32) bytessent; }    // total bytes sent, including 12-byte RTP headers
aligned(8) class hintPacketsSent extends box('npck') {
   uint(32) packetssent; } // total packets sent
aligned(8) class hintBytesSent extends box('tpay') {
   uint(32) bytessent; }    // total bytes sent, not including RTP headers

aligned(8) class hintmaxrate extends box('maxr') {    // maximum data rate
   uint(32) period;         // in milliseconds
   uint(32) bytes; }        // max bytes sent in any period 'period' long
                            //  including RTP headers

aligned(8) class hintmediaBytesSent extends box('dmed') {
   uint(64) bytessent; }    // total bytes sent from media tracks
aligned(8) class hintimmediateBytesSent extends box('dimm') {
   uint(64) bytessent; }    // total bytes sent immediate mode
aligned(8) class hintrepeatedBytesSent extends box('drep') {
   uint(64) bytessent; }    // total bytes in repeated packets

aligned(8) class hintminrelativetime extends box('tmin') {
   int(32)     time; }      // smallest relative transmission time, milliseconds
aligned(8) class hintmaxrelativetime extends box('tmax') {
   int(32)     time; }      // largest relative transmission time, milliseconds

aligned(8) class hintlargestpacket extends box('pmax') {
   uint(32) bytes; }        // largest packet sent, including RTP header
aligned(8) class hintlongestpacket extends box('dmax') {
   uint(32) time; }         // longest packet duration, milliseconds

aligned(8) class hintpayloadID extends box('payt') {
   uint(32) payloadID;      // payload ID used in RTP packets
   uint(8)     count;
   char     rtpmap_string[count]; }
```

NOTE Not all these sub-boxes may be present, and that there may be multiple 'maxr' boxes, covering different periods.

9.2 ALC/LCT and FLUTE Hint Track Format

9.2.1 Introduction

The file format supports multicast/broadcast delivery of files with FEC protection. Files to be delivered are stored as items in a container file (defined by the file format) and the meta box is amended with information on how the files are partitioned into source symbols. For each source block of a FEC encoding, additional parity symbols can be pre-computed and stored as FEC reservoir items. The partitioning depends on the FEC scheme, the target packet size, and the desired FEC overhead. Pre-composed source symbols can be stored as File reservoir items to minimize duplicate information in the container file especially with MBMS-FEC. The actual transmission is governed by hint tracks that contain server instructions that facilitate the encapsulation of source and FEC symbols into packets.

FD hint tracks have been designed for the ALC/LCT (Asynchronous Layered Coding/Layered Coding Transport) and FLUTE (File Delivery over Unidirectional Transport) protocols. LCT provides transport level support for reliable content delivery and stream delivery protocols. ALC is a protocol instantiation of the LCT building block, and it serves as a base protocol for massively scalable reliable multicast distribution of arbitrary binary objects. FLUTE builds on top of ALC/LCT and defines a protocol for unidirectional delivery of files.

FLUTE defines a File Delivery Table (FDT), which carries metadata associated with the files delivered in the ALC/LCT session, and provides mechanisms for in-band delivery and updates of FDT. In contrast, ALC/LCT relies on other means for out-of-band delivery of file metadata, e.g., an electronic service guide that is normally delivered to clients well in advance of the ALC/LCT session combined with update fragments that can be sent during the ALC/LCT session.

File partitionings and FEC reservoirs can be used independently of FD hint tracks and vice versa. The former aid the design of hint tracks and allow alternative hint tracks, e.g., with different FEC overheads, to re-use the same FEC symbols. They also provide means to access source symbols and additional FEC symbols independently for post-delivery repair, which may be performed over ALC/LCT or FLUTE or out-of-band via another protocol. In order to reduce complexity when a server follows hint track instructions, hint tracks refer directly to data ranges of items or data copied into hint samples.

It is recommended that a server sends a different set of FEC symbols for each retransmission of a file.

The syntax for using the meta box as a container file for source files is defined in 8.11, partitions, file and FEC reservoirs are defined in 8.13, while the syntax for FD hint tracks is defined in 9.2.

9.2.2 Design principles

The support for file delivery is designed to optimize the server transmission process by enabling ALC/LCT or FLUTE servers to follow simple instructions. It is enough to follow one pre-defined sequence of instructions per channel in order to transmit one session. The file format enables storage of pre-computed source blocks and symbol partitionings, i.e., files may be partitioned into symbols which fit an intended packet size, and pre-computing a certain amount of FEC-symbols that also can be used for post-session repair. The file format also allows storage of alternative ALC/LCT or FLUTE transmission session instructions that may lead to equivalent end results. Such alternatives may be intended for different channel conditions because of higher FEC protection or even by using different error correction schemes. Alternative sessions can refer to a common set of symbols. The hint tracks are flexible and can be used to compose FDT fragments and interleaving of such fragments within the actual object transmission. Several hint tracks can be combined into one or more sessions involving simultaneous transmission over multiple channels.

It is important to make a difference between the definition of sessions for transmission and the scheduling of such sessions. ALC/LCT and FLUTE server files only address optimization of the server transmission process. In order to ensure maximal usage and flexibility of such pre-defined sessions, all details regarding scheduling addresses, etc. are kept outside the definition of the file format. External scheduling applications decide such details, which are not important for optimizing transmission sessions per se. In particular, the following information is out-of-scope of the file format: time scheduling, target addresses and ports, source addresses and ports, and so-called Transmission Session Identifiers (TSI).

The sample numbers associated with the samples of a file delivery hint track provide a numbered sequence. Hint track sample times provide send times of ALC/LCT or FLUTE packets for a default bitrate. Depending on the actual transmission bitrate, an ALC/LCT or FLUTE server may apply linear time scaling. Sample times may simplify the scheduling process, but it is up to the server to send ALC/LCT or FLUTE packets in a timely manner.

A schematic picture of a file containing three alternative hint tracks with different FEC overhead for a source file is provided in Figure 6. In this example, each source block consists of only one sub-block.

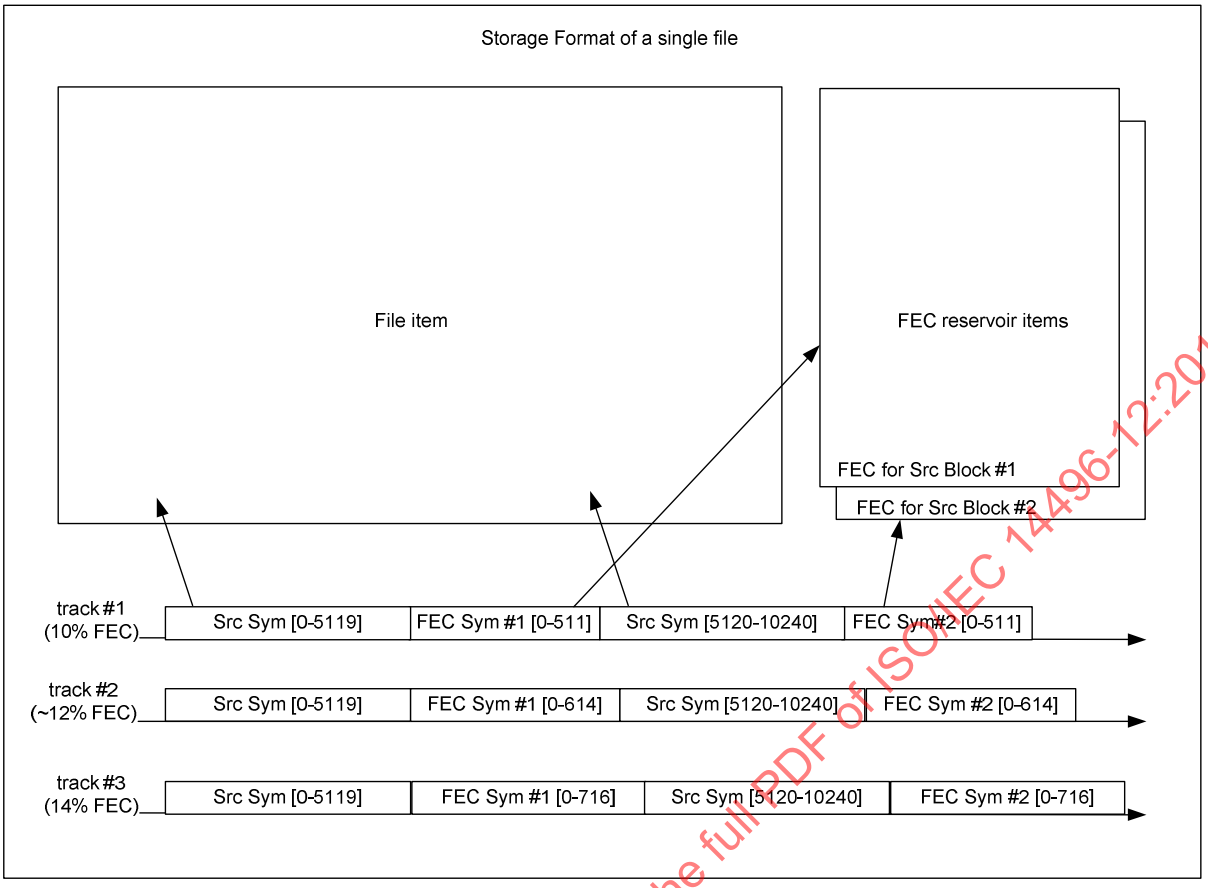

Figure 4 — Different FEC overheads of a source file provided by alternative hint tracks.

The source file in the above figure is partitioned into 2 source blocks containing symbols of a fixed size. FEC redundancy symbols are calculated for both source blocks and stored as FEC reservoir items. As the hint tracks reference the same items in the file there is no duplication of information. The original source symbols and FEC reservoirs can also be used by repair servers that don't use hint tracks.

9.2.3 Sample Description Format

9.2.3.1 Definition

FD hint tracks are tracks with `handler_type 'hint'` and with the entry-format `'fdp '` in the sample description box. The FD hint sample entry is contained in the sample description box (`'stsd'`).

9.2.3.2 Syntax

```
class FDHintSampleEntry() extends SampleEntry ('fdp ') {
   unsigned int(16)  hinttrackversion = 1;
   unsigned int(16)  highestcompatibleversion = 1;
   unsigned int(16)  partition_entry_ID;
   unsigned int(16)  FEC_overhead;
   Box               additionaldata[];   //optional
}
```

9.2.3.3 Semantics

`partition_entry_ID` indicates the partition entry in the FD item information box. A zero value indicates that no partition entry is associated with this sample entry, e.g., for FDT. If the corresponding FD hint track contains only overhead data this value should indicate the partition entry whose overhead data is in question.

`FEC_overhead` is a fixed 8.8 value indicating the percentage protection overhead used by the hint sample(s). The intention of providing this value is to provide characteristics to help a server select a session group (and corresponding FD hint tracks). If the corresponding FD hint track contains only overhead data this value should indicate the protection overhead achieved by using all FD hint tracks in a session group up to the FD hint track in question.

The `hinttrackversion` and `highestcompatibleversion` fields have the same interpretation as in the RTP hint sample entry described in 9.1.2. As additional data a time scale entry box may be provided. If not provided, there is no indication given on timing of packets.

File entries needed for an FDT or an electronic service guide can be created by observing all sample entries of a hint track and the corresponding item information boxes of the items referenced by the above partition entry IDs. No sample entries shall be included in the hint track if they are not referenced by any sample.

9.2.4 Sample Format

9.2.4.1 Sample Container

Each FD sample in the hint track will generate one or more FD packets.

Each sample contains two areas: the instructions to compose the packets, and any extra data needed when sending those packets (e.g., encoding symbols that are copied into the sample instead of residing in items for source files or FEC). Note that the size of the sample is known from the sample size table.

```
aligned(8) class FDsample extends Box('fdsa') {
   FDPacketBox    packetbox[]
   ExtraDataBox   extradata;   //optional
}
```

Sample numbers of FD samples define the order they shall be processed by the server. Likewise, FD packet boxes in each FD sample should appear in the order they shall be processed. If the time scale entry box is present in the FD hint sample entry, then sample times are defined and provide relative send times of packets for a default bitrate. Depending on the actual transmission bitrate, a server may apply linear time scaling. Sample times may simplify the scheduling process, but it is up to the server to send packets in a timely manner.

9.2.4.2 Packet Entry Format

Each packet in the FD sample has the following structure (References: RFC 3926, 3450, 3451):

```
aligned(8) class FDpacketBox extends Box('fdpa') {
   LCTheaderTemplate     LCT_header_info;
   unsigned int(16)      entrycount1;
   LCTheaderExtension    header_extension_constructors[ entrycount1 ];
   unsigned int(16)      entrycount2;
   dataentry             packet_constructors[ entrycount2 ];
}
```

The LCT header info contains LCT header templates for the current FD packet. Header extension constructors are structures which are used for constructing the LCT header extensions. Packet constructors are used for constructing the FEC payload ID and the source symbols in an FD packet.

9.2.4.3 LCT Header Template Format

The LCT header template is defined as follows:

```
aligned(8) class LCTheaderTemplate {
   unsigned int(1)   sender_current_time_present;
   unsigned int(1)   expected_residual_time_present;
   unsigned int(1)   session_close_bit;
   unsigned int(1)   object_close_bit;
   unsigned int(4)   reserved;
   unsigned int(16)  transport_object_identifier;
}
```

It can be used by a server to form an LCT header for a packet. Note that some parts of the header depend on the server policy and are not included in the template. Some field lengths also depend on the LCT header bits assigned by the server. The server may also need to change the value of the Transport Object Identifier (TOI).

9.2.4.4 LCT Header Extension Constructor Format

The LCT header extension constructor format is defined as follows:

```
aligned(8) class LCTheaderextension {
   unsigned int(8) header_extension_type;
   if (header_extension_type > 127) {
      unsigned int(8) content[3];
   }
   else {
   unsigned int(8) length;
   if (length > 0) {
      unsigned int(8) content[(length*4) - 2];
   }
}
```

A positive value of the length field specifies the length of the constructor content in multiples of 32 bit words. A zero value means that the header is generated by the server.

The usage and rules for LCT header extensions are defined in RFC 3451 (LCT RFC). The `header_extension_type` contains the LCT Header Extension Type (HET) value.

HET values between 0 and 127 are used for variable-length (multiple 32-bit word) extensions. HET values between 128 and 255 are used for fixed length (one 32-bit word) extensions. If the `header_extension_type` is smaller than 128, then the length field corresponds to the LCT Header Extension Length (HEL) as defined in RFC 3451. The content field always corresponds to the Header Extension Content (HEC).

NOTE A server can identify packets including FDT by observing whether EXT_FDT (`header_extension_type == 192`) is present.

9.2.4.5 Packet Constructor Format

There are various forms of the constructor. Each constructor is 16 bytes in order to make iteration easier. The first byte is a union discriminator. The packet constructors are used to include FEC payload ID as well as source and parity symbols in an FD packet.

```
aligned(8) class FDconstructor(type) {
   unsigned int(8)   constructor_type = type;
}

aligned(8) class FDnoopconstructor extends FDconstructor(0)
{
   unsigned int(8)   pad[15];
}

aligned(8) class FDimmediateconstructor extends FDconstructor(1)
{
   unsigned int(8)   count;
   unsigned int(8)   data[count];
   unsigned int(8)   pad[14 - count];
}

aligned(8) class FDsampleconstructor extends FDconstructor(2)
{
   signed int(8)     trackrefindex;
   unsigned int(16)  length;
   unsigned int(32)  samplenumber;
   unsigned int(32)  sampleoffset;
   unsigned int(16)  bytesperblock = 1;
   unsigned int(16)  samplesperblock = 1;
}

aligned(8) class FDitemconstructor extends FDconstructor(3)
{
   unsigned int(16)  item_ID;
   unsigned int(16)  extent_index;
   unsigned int(64)  data_offset;   //offset in byte within extent
   unsigned int(24)  data_length;   //non-zero length in byte within extent or
                                    //if (data_length==0) rest of extent
}

aligned(8) class FDxmlboxconstructor extends FDconstructor(4)
{
   unsigned int(64)  data_offset; //offset in byte within XMLBox or BinaryXMLBox
   unsigned int(32)  data_length;
   unsigned int(24)  reserved;
}
```

9.2.4.6 Extra Data Box

Each sample of an FD hint track may include extra data stored in an extra data box:

```
aligned(8) class ExtraDataBox extends Box('extr') {
   FECInformationBox  feci;
   bit(8)   extradata[];
}
```

9.2.4.7 FEC Information Box

9.2.4.7.1 Definition

Box Type: 'feci'
Container: Extra Data Box ('extr')
Mandatory: No
Quantity: Zero or One

The FEC Information box stores FEC encoding ID, FEC instance ID and FEC payload ID which are needed when sending an FD packet.

9.2.4.7.2 Syntax

```
aligned(8) class FECInformationBox extends Box('feci') {
   unsigned int(8)   FEC_encoding_ID;
   unsigned int(16)  FEC_instance_ID;
   unsigned int(16)  source_block_number;
   unsigned int(16)  encoding_symbol_ID;
}
```

9.2.4.7.3 Semantics

`FEC_encoding_ID` identifies the FEC encoding scheme and is subject to IANA registration (see RFC 5052), in which (i) value zero corresponds to the "Compact No-Code FEC scheme" also known as "Null-FEC" (RFC 3695); (ii) value one corresponds to the “MBMS FEC” (3GPP TS 26.346); (iii) for values in the range of 0 to 127, inclusive, the FEC scheme is Fully-Specified, whereas for values in the range of 128 to 255, inclusive, the FEC scheme is Under-Specified.

`FEC_instance_ID` provides a more specific identification of the FEC encoder being used for an Under-Specified FEC scheme. This value should be set to zero for Fully-Specified FEC schemes and shall be ignored when parsing a file with FEC_encoding_ID in the range of 0 to 127, inclusive. FEC_instance_ID is scoped by the FEC_encoding_ID. See RFC 5052 for further details.

`source_block_number` identifies from which source block of the object the encoding symbol(s) in the FD packet are generated.

`encoding_symbol_ID` identifies which specific encoding symbol(s) generated from the source block are carried in the FD packet.

9.3 MPEG-2 Transport Hint Track Format

9.3.1 Introduction

MPEG-2 TS (Transport Stream) is a stream multiplex which can carry one or more programs, consisting of audio, video and other media. The file format supports the storage of MPEG-2 TS in a hint track. An MPEG-2 TS hint track can be used for both storage of received TS packets (as a reception hint track), and as a server hint track used for the generation of an MPEG-2 TS.

The MPEG-2 TS hint track definition supports so-called “precomputed hints”. Precomputed hints make no use of including data by reference from other tracks, but rather MPEG-2 TS packets are stored as such. This allows reusing the MPEG-2 TS packets stored in a separate file. Furthermore, precomputed hints facilitate simple recording operation.

In addition to precomputed hint samples, it is possible to include media data by reference to media tracks into hint samples. Conversion of a received transport stream to media tracks would allow existing players compliant with earlier versions of the ISO base media file format to process recorded files as long as the media formats are also supported. Storing the original transport headers retains valuable information for error concealment and the reconstruction of the original transport stream.

9.3.2 Design Principles

The design principles of the MPEG-2 TS Hint Track Format are as follows.

A sequence of samples in an MPEG-2 TS Hint Track is a set of precomputed and constructed MPEG-2 TS packets. Precomputed packets are TS packets which are stored unchanged in the case of reception or will be sent as is. This is especially important where data cannot be de-multiplexed and elementary streams cannot be created – e.g. when the transport stream is encrypted and is not allowed to be stored decrypted. Therefore, it is necessary to be able to store the MPEG-2 TS as such in a hint track. Constructed packets use the same approach as RTP hint tracks, i.e., the sample contains instructions for a streaming server to construct the packet. The actual media data is contained in other tracks. A track reference of type ‘hint’ is used.

9.3.2.1 Reusing existing Transport Streams

It was desired to reuse existing TS instances and therefore an additional mechanism exists to cover a wide variety of existing TS recordings. These recordings may consist not only of TS packets but have preceding or trailing data with each TS packet. A specific case for preceding data is a 4-byte timestamp in front of each TS packet to remove the jitter of a transmission system. A specific case for trailing data is the addition of FEC when a TS packet is transmitted over an error-prone channel.

9.3.2.2 Timing

MPEG-2 TS defines a single clock for each program, running at 27MHz, which sampling value is transported as PCRs in the TS for clock recovery. The timescale of MPEG-2 TS Hint Tracks is recommended to be 90000, or an integer division or multiple thereof.

The decoding time of a sample in a MPEG-2 TS Hint Track is the reception/transmission time of the first bit of that packet or packet group which is recommended to be derived from the PCR timestamps of the TS, since if the PCR times are used, piece-wise linearity can be assumed and the 'stts' table compacts sensibly. The optional 'tsti' box in the sample description can be used to signal whether reception timing with or without clock recovery was used when the hint track is a reception hint track. In the case of a server hint track PCR timing is assumed.

NOTE: When there are multiple packets in a sample, they cannot be given independent transmission time offsets.

9.3.2.3 Packet Grouping

The sample format for MPEG-2 Transport Stream Hint Tracks allows multiple TS packets in one sample. Specific applications, such as some IPTV applications, convey TS packets in an RTP stream. Only one reception timestamp can be derived for all TS packets carried in one RTP packet. Another application for storing multiple TS packets in a sample is SPTSs, where a sample contains all the TS packets for a GoP. In this case every sample is a random access point.

Note that random-access to every TS packet is not possible by the means of the file format if multiple TS packets per sample are used.

In the case of an MPTS only one packet per sample should be used. This facilitates the use of the sample group mechanism on a per-packet basis.

9.3.2.4 Random-access points

A sync sample is a point at which processing of a track may begin without error. Both MPTS and SPTS are supported by MPEG-2 TS Hint Tracks, however a random access point, marked as a sync sample, is normally only defined for SPTS, where it specifies the beginning of a packet that contains the first byte of an independently decodable media access unit (e.g. MPEG-2 video I-frames or MPEG-4 AVC IDR pictures) of a stream that uses differential coding. For MPTS, the sync sample table would normally be present but empty, indicating that there is no point in the track at which processing of the entire track may begin without error. It is recommended that the PSI/SI be in the Sample Description so that true random-access with just the media data is possible.

Note that in the case of an MPTS, the sync sample table is present but empty (which means essentially that no sample is a sync sample).

Note also that in case of an SPTS, samples including multiple TS packets should have a sync point (e.g. GoP boundary) at the start of a sample. The sync sample table then marks the samples the sync points (e.g. the start of GoPs); if the sync sample table is absent, all the samples are sync points. If the sync sample table is present but empty, the sync sample positions are unknown and may be not at the start of samples.

NOTE: An application searching for a key frame can start reading at that location, but in general it also has to read further MPEG-2 TS packets (regarding the file format these are subsequent samples) so that the decoder can decode a complete frame.

9.3.2.5 Application as a Reception Hint Track

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

NOTE 1: 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.

Reception hint tracks have the same structure as hint tracks for servers.

The format of the reception hint samples is indicated by the sample description for the reception hint track. Each protocol has its own reception hint sample format and name.

NOTE 2: 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 3: As with server hint tracks, the sample formats of reception hint tracks may enable construction of packets by pulling data out of other tracks by reference. These other tracks may be hint tracks or media 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, i.e., indirect data referencing is disallowed. 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.

If received data is extracted to media tracks, the de-hinting process must ensure that the media streams are valid, i.e. the streams must be error-free (which requires e.g. error concealment).

A sample with a size of zero is permitted in reception hint tracks, and such samples may be ignored.

9.3.3 Sample Description Format

9.3.3.1 Introduction

The sample description for an MPEG2-TS reception hint track contains all static metadata that describe the stream or a portion thereof, especially the PSI/SI tables. MPEG-2 TS reception hint tracks use an entry-format in the sample description of 'rm2t' (which indicates *MPEG-2 Transport Stream*). The entry-format for MPEG2-TS server hint tracks is 'sm2t'.

The static metadata documents e.g. PSI/SI tables. The presence of static metadata is optional. When present, the static metadata shall be valid for the MPEG2-TS packets it describes. Consequently, if a piece of static metadata changes in the stream, a new sample entry is needed for the first sample at or after the change. If static metadata is not present in the sample entry, structures, such as PSI/SI tables, stored in the MPEG2-TS packets are valid and the stream must be scanned in order to find out which values of static metadata are valid for a particular sample.

9.3.3.2 Syntax

```
class MPEG2TSReceptionSampleEntry extends MPEG2TSSampleEntry(`rm2t´) {
}


class MPEG2TSServerSampleEntry extends MPEG2TSSampleEntry(`sm2t´) {
}
```

```
class MPEG2TSSampleEntry(name) extends HintSampleEntry(name) {
   uint(16) hinttrackversion = 1;
   uint(16) highestcompatibleversion = 1;
   uint(8)  precedingbyteslen;
   uint(8)  trailingbyteslen;
   uint(1)  precomputed_only_flag;
   uint(7)  reserved;
   box      additionaldata[];
}
```

9.3.3.3 Semantics

`hinttrackversion` is currently 1; the `highestcompatibleversion` field specifies the oldest version with which this track is backward-compatible.

`precedingbyteslen` indicates the number of bytes that are preceding each MPEG2-TS packet (which may e.g. be a time-code from an external recording device).

`trailingbyteslen` indicates the number of bytes that are at the end of each MPEG2-TS packet (which may e.g. contain checksums or other data that was added by a recording device).

`precomputed_only_flag` indicates whether the associated samples are purely precomputed if set to 1;

`additionaldata` is a set of boxes. This set can contain boxes that describe one common version of the PSI/SI tables by means of the 'tPAT' box or the 'tPMT' box or other data, e.g. boxes that are only valid for a sample (which contains multiple packets) and describe the initial conditions of the STC or boxes that define the content of the preceding or trailing data. There shall be at most one of each of PATBox, TSTimingBox, InitialSampleTimeBox present within additionaldata

The following optional boxes for `additionaldata` are defined:

```
aligned(8) class PATBox() extends Box('tPAT') {
   uint(3)     reserved;
   uint(13)    PID;
   uint(8)     sectiondata[];
}


aligned(8) class PMTBox() extends Box('tPMT') {
   uint(3)     reserved;
   uint(13)    PID;
   uint(8)     sectiondata[];
}

aligned(8) class ODBox () extends Box ('tOD ') {
   uint(3)     reserved;
   uint(13)    PID;
   uint(8)     sectiondata[];
}

aligned(8) class TSTimingBox() extends Box('tsti') {
   uint(1)     timing_derivation_method;
   uint(2)     reserved;
   uint(13)    PID;
}

aligned(8) class InitialSampleTimeBox() extends Box('istm') {
   uint(32) initialsampletime;
   uint(32) reserved;
}
```

The 'tPAT' box contains the section data of the PAT and each 'tPMT' box contains the section data of one of the PMTs.

In the case of an SPTS, it is strongly recommended that the 'tPMT' box is present in the `additionaldata`. If the PMT is not present in the sample data, then it shall be present in the `additionaldata`. If the 'tPMT' box is present, it shall be the PMT for the program contained in the sample data (although the recorded stream may contain other programs and be an MPTS).

`PID` is the PID of the MPEG2-TS packets from which the data was extracted. In the case of the 'tPAT' box this value is always 0.

`sectiondata` extends to the end of the box and is the complete MPEG2-TS table, containing the concatenated sections, of an identical version number.

`initialsampletime` specifies the initial value of the sample times in case the sample times do not start from 0. Unlike media tracks, MPEG-2 TS hint track usually have sample times not starting from 0, e.g., PCR times and reception times. Since 'stts' only stores the delta between sample times, this field is required for reconstructing the original sample times:

```
OriginalSampleTime(n) = initialsampletime + STTS(n).
```

In case PCR times are used for sample times, the reconstructed sample time can be used to initialize the STC when the sample is randomly accessed. Note that this field may need to be updated after editing.

`timing_derivation_method` is a flag which specifies the method which was used to set the sample time for a given PID. The values for `timing_derivation_method` are as follows:

`0x0` reception time: the sample timing is derived from the reception time. It is not guaranteed that the STC was recovered for derivation of the reception time.

`0x1` piecewise linearity between PCRs: the sample time is derived from a reconstructed STC for this program. Piecewise linearity between adjacent PCRs is assumed and all TS packets in the samples have a constant duration in this range.

9.3.4 Sample Format

Each sample of an MPEG-2 TS Hint track consists of a set of

- pre-computed packets: one or more MPEG-2 TS packets with the associated headers and trailers
- constructed packets: instructions to compose one or more MPEG2-TS packets with the associated headers and trailers by pointing to data of another track.

Note that each MPEG-2 TS packet in the sample may be preceded with a preheader (`precedingbytes`), or followed by a posttrailer (`trailingbytes`), as detailed in the Sample Description Format. The size of the preheader and the posttrailer are specified by `precedingbyteslen` and `trailingbyteslen`, respectively, in the sample description to allow compact sample tables with fewer chunks.

It is possible for a mixture of pre-computed and constructed samples to occur in the same track. If padding of the transport stream packet is required, this can be accomplished with the `adaptation_field` or explicitly by using the `MPEG2TSImmediateConstructor` as appropriate.

NOTE: The number of MPEG-2 TS packets in the sample can be derived from the sample size table directly if the sample consists of pre-computed packets only, which is a conclusion if the `precomputed_only_flag` in the sample entry is set. The number of MPEG-2 TS packets in the sample may be variable or restricted, e.g. extensions of this file format may define a sample to contain exactly one packet.

9.3.4.1 Syntax

```
// Constructor format
aligned(8) abstract class MPEG2TSConstructor (uint(8) type) {
   uint(8)        constructor_type = type;
}
```

```
aligned(8) class MPEG2TSImmediateConstructor
   extends MPEG2TSConstructor(1) {
   uint(8)        immediatedatalen;
   uint(8)        data[immediatedatalen];
}

aligned(8) class MPEG2TSSampleConstructor
   extends MPEG2TSConstructor(2) {
   uint(8)     sampledatalen;
   uint(16)    trackrefindex;
   uint(32)    samplenumber;
   uint(32)    sampleoffset;
}

// Packet format
aligned(8) class MPEG2TSPacketRepresentation {
   uint(8)  precedingbytes[precedingbyteslen];
   uint(8)  sync_byte;
   if (sync_byte == 0x47) {
      uint(8)     packet[187];
   } else if (sync_byte == 0x00) {
      uint(8)     headerdatalen;
      uint(4)     reserved;
      uint(4)     num_constructors;
      bit(1)      transport_error_indicator;
      bit(1)      payload_unit_start_indicator;
      bit(1)      transport_priority;
      bit(13)     PID;
      bit(2)      transport_scrambling_control;
      bit(2)      adaptation_field_control;
      bit(4)      continuity_counter;
      if (adaptation_field_control == '10' ||
          adaptation_field_control == '11') {
         uint(8)     adaptation_field[headerdatalen-3];
      }
      MPEG2TSConstructor   constructors[num_constructors];
   }
   uint(8)   trailingbytes[trailingbyteslen];
}

// Sample format
aligned(8) class MPEG2TSSample {
   MPEG2TSPacket  packet[];
}
```

9.3.4.2 Semantics

`precedingbytes` contains any extra data preceding the packet, typically provided by the recording device. For example, this may include a timestamp.

`sync_byte`: if this value is 0x47, then the sample is a transport stream packet (a precomputed reception hint track sample), with the remaining bytes following in the field `packet`. If this value is 0x00, it indicates that the associated sample points to a track indexed by `trackrefindex` in the track reference box with reference type 'hint'. All other values are currently reserved. When the packet data is actually put into a streaming channel, the value shall always be set to 0x47

`trackrefindex` indexes in the track reference box with reference type 'hint' to indicate with which media track the current sample is associated. The `samplenumber` and `sampleoffset` fields in the `MPEG2TSSampleConstructor` point into this media track. The `trackrefindex` starts from value 1. The value 0 is reserved for future use.

`packet`: The MPEG-2 TS packet, apart from the sync byte (0x47).

The `MPEG2TSConstructor` array is a collection of one or more constructor entries, to allow for multiple access units in one transport stream packet. An `MPEG2TSImmediateConstructor` can contain, amongst others, the PES header. An `MPEG2TSSampleConstructor` references data in the associated media track. The sum of `headerdatalen` and the `datalen` fields of all constructors of an `MPEG2TSPacket` must be equal to the length of the transport stream packet being constructed, minus 1 byte, which is 187.

`trailingbytes` contains any extra data following the packet. For example, this may include a checksum.

`samplenumber` indicates the sample within the referred track contained in the packet and `sampleoffset` indicates the starting byte position of the referred media sample contained in the packet of which `sampledatalen` bytes are included. `sampleoffset` starts from value 0.

`immediatedatalen` indicates the number of bytes within the field `data` that are included in the sample rather than data being included into the sample by reference to a media track.

`headerdatalen` indicates the length of the TS packet header (without the sync byte) in bytes. This field has the value 3 if the `adaptation_field` is not present or the value `(adaptation_field_length+3)`, where `adaptation_field_length` is the first octet of the structure `adaptation_field` as defined in ISO/IEC 13818-1.

Neither the format of `precedingbytes` nor `trailingbytes` are defined by this specification.

The remaining fields (`transport_error_indicator`, `payload_unit_start_indicator`, `transport_priority`, `PID`, `transport_scrambling_control`, `adaptation_field_control`, `continuity_counter`, `adaptation_field`) of the sample structure contain a copy of the packet header of the TS packet, as defined in ISO/IEC 13818-1.

9.3.5 Protected MPEG 2 Transport Stream Hint Track

9.3.5.1 Introduction

This Subclause defines a mechanism for marking media streams as protected. This works by changing the four character code of the SampleEntry, and appending boxes containing both details of the protection mechanism and the original four character code. However, in this case the track is not protected; it is an 'in the clear' hint track which contains protected data. This Subclause describes how hint tracks should be marked as carrying protected data, using a similar mechanism, and utilizing the same boxes.

9.3.5.2 Syntax

```
class ProtectedMPEG2TransportStreamSampleEntry
   extends MPEG2TransportStreamSampleEntry('pm2t') {
   ProtectionSchemeInfoBox    SchemeInformation;
}
```

9.3.5.3 Semantics

The `SchemeInformation` ("sinf") box (defined in 8.12) shall contain details of the protection scheme applied. This shall include the `OriginalFormatBox` which shall contain the original sample entry type of the MPEG-2 Transport StreamSampleEntry box.

9.4 RTP, RTCP, SRTP and SRTCP Reception Hint Tracks

9.4.1 RTP Reception Hint Track

9.4.1.1 Introduction

This Subclause specifies the reception hint track format for the real-time transport protocol (RTP), as defined in IETF RFC 3550.

RTP is used for real-time media transport over the Internet Protocol. Each RTP stream carries one media type, and one RTP reception hint track carries one RTP stream. Hence, recording of an audio-visual program results into at least two RTP reception hint tracks.

The design of the RTP reception hint track format follows as much as possible the design of the RTP server hint track format. This design should ensure that RTP packet transmission operates very similarly regardless whether it is based on RTP reception hint tracks or RTP server hint tracks. Furthermore, the number of new data structures in the file format was consequently kept as small as possible.

The format of the RTP reception hint tracks allow storing of the packet payloads in the hint samples, or converting the RTP packet payloads to media samples and including them by reference to the hint samples, or combining both approaches. As noted earlier, 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 also supported. Storing the original RTP headers retains valuable information for error concealment and the reconstruction of the original RTP stream. It is noted that the conversion of packet payloads to media samples may happen "off-line" after recording of the streams in precomputed RTP reception hint tracks has been completed.

9.4.1.2 Sample Description Format

The entry-format in the sample description for the RTP reception hint tracks is 'rrtp'. The syntax of the sample entry is the same as for RTP server hint tracks having the entry-format 'rtp '.

```
class ReceivedRtpHintSampleEntry() extends SampleEntry (‘rrtp‘) {
   uint(16)    hinttrackversion = 1;
   uint(16)    highestcompatibleversion = 1;
   uint(32)    maxpacketsize;
   box         additionaldata[];
}
```

The entry-format identifier in the sample description of the RTP reception hint track is different from the entry-format in the sample description of the RTP server hint track, in order to avoid using an RTP reception hint track that contains errors as a valid server hint track.

The `additionaldata` set of boxes may include the timescale entry ('tims') and time offset ('tsro') boxes. Moreover, the `additionaldata` may contain a timestamp synchrony box.

The timescale entry box (`‘tims’`) shall be present and the value of timescale shall be set to match the clock frequency of the RTP timestamps of the stream captured in the reception hint track.

The time offset box (`‘tsro’`) may be present. If the time offset box is not present, the value of the field `offset` is inferred to be equal to 0. The value of the field `offset` is used for the derivation of the RTP timestamp, as specified in 9.4.1.4.

RTP timestamps typically do not start from zero, especially if an RTP receiver 'tunes' into a stream. The time offset box should therefore be present in RTP reception hint tracks and the value of `offset` in the time offset box should be set equal to the first RTP timestamp of the RTP stream in reception order.

Zero or one `timestampsynchrony` boxes may be present in the `additionaldata` of the sample entry for a RTP reception hint track. If a `timestampsynchrony` box is not present, the value of `timestamp_sync` is inferred to be equal to 0.

```
class timestampsynchrony() extends Box(‘tssy’) {
   unsigned int(6) reserved;
   unsigned int(2) timestamp_sync;
}
```

`timestamp_sync` equal to 0 indicates that the RTP timestamps of the present RTP reception hint track derived from the Formula in 9.4.1.4 may or may not be synchronized with RTP timestamps of other RTP reception hint tracks.

`timestamp_sync` equal to 1 indicates that the RTP timestamps of the present RTP reception hint track derived from the Formula in 9.4.1.4 reflect the received RTP timestamps exactly (without corrected synchronization to any other RTP reception hint track).

`timestamp_sync` equal to 2 indicates that RTP timestamps of the present RTP reception hint track derived from the Formula in 9.4.1.4 are synchronized with RTP timestamps of other RTP reception hint tracks.

When `timestamp_sync` is equal to 0 or 1, a player should correct the inter-stream synchronization using stored RTCP sender reports. When `timestamp_sync` is equal to 2, the media contained in the RTP reception hint tracks can be played out synchronously according to the reconstructed RTP timestamps without synchronization correction using RTCP Sender Reports. If it is expected that the RTP reception hint track will be used for re-sending the recorded RTP stream, it is recommended that `timestamp_sync` be set equal to 0 or 1, because the stored RTCP sender reports can be reused.

`timestamp_sync` equal to 3 is reserved.

The value of `timestamp_sync` shall be identical for all RTP reception hint tracks present in a file.

When RTCP is also stored, using an RTCP hint track, the timestamp relationship between the RTP and RTCP hint tracks can only be maintained if the RTP timestamps are anchored by using a set time offset ('tsro') in the RTP track, and hence the time offset is mandatory if RTCP is stored in an RTCP hint track.

Zero or one `ReceivedSsrcBox` identified with the four-character code 'rssr' shall be present in the `additionaldata` of a sample descriptor entry of a RTP reception hint track:

```
class ReceivedSsrcBox extends Box('rssr') {
   unsigned int(32)  SSRC
}
```

The `SSRC` value must equal the SSRC value in the header of all recorded SRTP packets described by the sample description.

9.4.1.3 Sample Format

The sample format of RTP reception hint tracks is identical to the syntax of the sample format of the RTP server hint tracks. Each sample in the reception hint track represents one or more received RTP packets. If media frames are not both fragmented and interleaved in an RTP stream, it is recommended that each sample represents all received RTP packets that have the same RTP timestamp, i.e., consecutive packets in RTP sequence number order with a common RTP timestamp.

Each RTP reception hint sample contains two areas: the instructions to compose the packet, and any extra data needed for composing the packet, such as a copy of the packet payload. Note that the size of the sample is known from the sample size table.

Since the reception time for the packets may vary, this variation can be signalled for each packet as specified subsequently.

A sample with a size of zero is permitted in reception hint tracks, and such samples may be ignored.

9.4.1.4 Packet Entry Format

Each packet in the packet entry table has same structure as for server (transmission) hint tracks, in 9.1.3.1.

Where `i` is the sample number of a sample, the sum of the sample time DT(i) as specified in 8.6.1.2 and `relative_time` indicates the reception time of the packet. The clock source for the reception time is undefined and may be, for instance, the wall clock of the receiver. If the range of reception times of a reception hint track overlaps entirely or partly with the range of reception times of another reception hint track, the clock sources for these hint tracks shall be the same.

It is recommended that receivers may use a constant value for `sample_delta` in the decoding time to sample box ('stts') as much as reasonable and smooth out packet scheduling and end-to-end delay variation by setting `relative_time` adaptively in stored reception hint samples. This arrangement of setting the values of `sample_delta` and `relative_time` can facilitate a compact decoding time to sample box. In this case timestamp_sync is set to 1, the sample durations are mostly constant, and the time offset ('tsro') is stored in the sample entry.

The values of `RTP_version`, `P_bit`, `X_bit`, `CSRC_count`, `M_bit`, `payload_type`, and `RTPsequenceseed` shall be set equal to the V, P, X, CC, M, PT and sequence number fields of the RTP packet captured in the sample.

The fields `bframe_flag` and `repeat_flag` are reserved in reception hint tracks and must be zero.

The semantics of `extra_flag` and `extra_information_length` are identical to those of specified for the RTP server hint tracks.

The following TLV boxes are specified: `rtphdrextTLV`, `rtpoffsetTLV`, `receivedCSRC`.

If the `X_bit` is set a single `rtphdrextTLV` box shall be present for storing the received RTP Header Extension.

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

`data` is the raw RTP Header Extension which is application-specific.

The syntax of the `rtpoffsetTLV` box is specified in 9.1.3.1.

`offset` indicates a 32-bit signed integer offset to the RTP timestamp of the received RTP packet. Let i be the sample number of a sample, DT(i) be equal to DT as specified in 8.6.1.2 for sample number i, `tsro.offset` be the value of offset in the 'tsro' box of the referred reception hint sample entry, and % be the modulo operation. The value of `offset` shall be such that the following Formula is true:

$$RTPtimestamp = (DT_i + tsro.offset + offset) \bmod 2^{32}$$

formula (1) RTP timestamp calculation

NOTE 1: When each reception hint sample represents all received RTP packets that have the same RTP timestamp, the value of `sample_delta` in the decoding time to sample box can be set to match the RTP timestamp. In other words, DT(i), as specified above, can be set equal to (the RTP timestamp – `tsro.offset` – `offset`) (assuming that the resulting value would be greater than or equal to 0). This is recommended.

NOTE 2: RTP timestamps do not necessarily increase as a function of RTP sequence number in all RTP streams, i.e., transmission order and playback order of packets may not be identical. For example, many video coding schemes allow bi-prediction from previous and succeeding pictures in playback order. As samples appear in tracks in their decoding order, i.e., in reception order in case of RTP reception hint tracks, `offset` in the `rtpoffsetTLV` box can be used to warp the RTP timestamp away from the sample time DT(i).

For the purpose of edits in Edit List Boxes, the composition time of a received RTP packet is inferred to be the sum of the sample time DT(i) and `offset` as specified above.

If the value of `CSRC_count` is not equal to zero, a `receivedCSRC` box may be present for storing the received CSRC header fields for each RTP packet. The `receivedCSRC` box is identified with the four-character code ‘rcsr’

```
aligned(8) class receivedCSRC extends Box('rcsr') {
   unsigned int(32)  CSRC[];  //to end of the box
}
```

The number of entries in `CSRC[]` equals the `CC` value of received SRTP packets. The n^{th} entry of `CSRC[]` shall equal the n^{th} CSRC value of the RTP packet header.

9.4.1.5 SDP information

Both movie and track SDP information may be present, as specified in 9.1.4.

9.4.2 RTCP Reception Hint Track

9.4.2.1 Introduction

This Subclause specifies the reception hint track format for the real-time control protocol (RTCP), defined in IETF RFC 3550.

RTCP is used for real-time transport of control information for an RTP session over the Internet Protocol. During streaming, each RTP stream typically has an accompanying RTCP stream that carries control information for the RTP stream. One RTCP reception hint track carries one RTCP stream and is associated to the corresponding RTP reception hint track through a track reference.

The format of the RTCP reception hint tracks allows the storage of RTCP Sender Reports in the hint samples.

The RTCP Sender Reports are of particular interest for stream recording, because they reflect the current status of the server, e.g., the relationship of the media timing (RTP timestamp of audio/video packets) to the server time (absolute time in NTP format). Knowledge of this relationship is also necessary for playback of recorded RTP reception hint tracks to be able to detect and correct clock drift and jitter.

The timestamp synchrony box as specified in 9.4.1.2 makes it possible to correct clock drift and jitter before playing a file, and therefore recording of RTCP streams is optional when timestamp_sync is equal to 2.

There is no server hint track equivalent for the RTCP reception hint track, since RTCP messages are generated on-the-fly during transmission.

9.4.2.2 General

There shall be zero or one RTCP reception hint track for each RTP reception hint track. An RTCP reception hint track shall contain a track reference box including a reference of type '`cdsc`' to the associated RTP reception hint track.

When i is the sample number of a sample, the sample time DT(i) as specified in 8.6.1.2 indicates the reception time of the packet. The clock source for the reception time shall be the same as for the associated RTP reception hint track. The value of `timescale` in the Media Header Box of an RTCP reception hint track shall be equal to the value of `timescale` in the media header box of the associated RTP reception hint track.

9.4.2.3 Sample Description Format

The entry-format in the sample description for the RTCP reception hint tracks is 'rtcp'. It is otherwise identical in structure to the sample entry format for RTP. There are no defined boxes for the `additionaldata` field.

9.4.2.4 Sample Format

9.4.2.4.1 Introduction

Each sample in the reception hint track represents one or more received RTCP packets. Each sample contains two areas: the raw RTCP packets and any extra data needed. Note that the size of the sample is known from the sample size table, and that the size of an RTCP packet is indicated within the packet itself (as documented in RFC 3550), as a count one less than the number of 32-bit words in that packet.

9.4.2.4.2 Syntax

```
aligned(8) class receivedRTCPpacket {
   unsigned int(8)      data[];
}

aligned(8) class receivedRTCPsample {
   unsigned int(16)  packetcount;
   unsigned int(16)  reserved;
   receivedRTCPpacket   packets[packetcount];
}
```

9.4.2.4.3 Semantics

`data` contains a raw RTCP packet including the RTCP report header, the 20-byte sender information block and any number of report blocks. Note that the size of each RTCP packet is known by parsing the 16-bit length field of the RTCP header.
`packetcount` indicates the number of received RTCP packets contained in the sample.
`packets` contains the received RTCP packets.

9.4.3 SRTP Reception Hint Track

9.4.3.1 Introduction

This Subclause specifies the reception hint track formats for the secure real-time transport protocol (SRTP), as defined in IETF RFC 3711.

SRTP is a secure extension of the real-time media transport (RTP) over the Internet Protocol. Each SRTP stream carries one media type, and one SRTP reception hint track carries one SRTP stream. Hence, recording of an audio-visual program results into at least two SRTP reception hint tracks.

The design of the SRTP reception hint track format follows the design of RTP reception hint tracks and reuses most of the framework provided by RTP reception hint tracks. The major difference between RTP and SRTP reception hint tracks is that the actual media payload is stored in an encrypted form for SRTP reception hint tracks, whereas it is unencrypted for RTP reception hint tracks. SRTP reception hint tracks provide additional boxes to store information necessary to decrypt encrypted content on playback. Additionally, all header fields of the SRTP packet header shall be stored with the payload, as this information is necessary to check the integrity of the received data. SRTP reception hint tracks are commonly used together with SRTCP reception hint tracks.

SRTP reception hint tracks may, for example, be used to store protected mobile TV content.

9.4.3.2 Sample Description Format

9.4.3.2.1 Sample Description Entry

The sample description format for SRTP reception hint tracks is identical to that for RTP reception hint tracks with the exception that the sample entry name is changed from ‘rrtp’ to ‘rsrp’ and that it may contain additional boxes:

```
class ReceivedSrtpHintSampleEntry() extends SampleEntry (‘rsrp‘) {
   uint(16)    hinttrackversion = 1;
   uint(16)    highestcompatibleversion = 1;
   uint(32)    maxpacketsize;
   box         additionaldata[];
}
```

Fields and boxes are identical to those of the `ReceivedRtpHintSampleEntry (‘rrtp‘)`. The `addtionaldata[]` of each sample description entry of a SRTP Reception Hint Track shall contain exactly one `ReceivedSsrc` Box (‘rssr’).

Additionally, the additionaldata[] may contain the Received Cryptographic Context ID box and the Rollover Counter box defined below. Furthermore, a SRTP Process Box shall also be included as one of the `additionaldata` boxes. As the content is stored encrypted, the integrity and the encryption algorithm fields in the SRTP Process box specify the algorithm that was applied to the received stream. An entry of four spaces ($20$20$20$20) may be used to indicate that the algorithm is defined by means outside the scope of this document.

9.4.3.2.2 Received Cryptographic Context ID Box

Zero or one `ReceivedCryptoContextIdBox`, identified with the four-character code ‘ccid’, may be present in the `additionaldata` of a sample descriptor entry of an SRTP reception hint track. Information to recover the cryptographic context for the received SRTP stream may be stored here.

```
aligned(8) class ReceivedCryptoContextIdBox extends Box (‘ccid’) {
   unsigned int(16)  destPort;
   unsigned int(8) ip_version;
   switch (ip_version) {
      case 4: // IPv4
         unsigned int(32)  destIP;
         break;
      case 6: // IPv6
         unsigned int(64)  destIP;
         break;
   }
}
```

The `destPort` and `destIP` parameters contain the port number and the IP address (as present in the received IPv4 or IPv6 packages), respectively, of the SRTP session via which the recorded SRTP packets were received. `ip_version` contains either 4 or 6 representing IPv4 or IPv6, respectively.

9.4.3.2.3 Rollover Counter Box

Zero or one `RolloverCounterBox`, identified with the four-character code ‘sroc’, may be present in the `additionaldata` of a sample descriptor entry of an SRTP reception hint track. Typically, the rollover counter value changes every 65536 SRTP package.

```
aligned(8) class RolloverCounterBox extends Box (‘sroc’) {
   unsigned int(32)  rollover_counter;
   }
```

The `rollover_counter` is a non-zero integer that gives the value of the ROC field for all associated received SRTP packets.

> NOTE: The rollover counter (ROC) is an element of the cryptographic context of a SRTP stream and depends on the absolute position of a packet in an RTP stream. Knowledge of the ROC value is necessary in order to decrypt a received SRTP packet. It is optional to use the rollover counter box as RFC 4771 defines as an optional mechanism to signal the ROC value explicitly in the authentication tag of a SRTP package.

9.4.3.3 Sample and Packet Entry Format

Both, sample format and packet Entry format for SRTP reception hint tracks are identical to those of RTP reception hint tracks, defined in 9.4.1.3 and 9.4.1.4. The packet payload is stored as received in the SRTP packets, i.e., all information received in the SRTP packet excluding the header or, in other words, the encrypted payload together with the key identifier (MKI) and the authentication tag.

If the value of `CSRC_count` is not equal to zero for a received SRTP packet, the `extra_data_tlv` corresponding to this `receivedSRTPpacket` shall contain exactly one `receivedCSRC` box ('`rcsr`').

9.4.4 SRTCP Reception Hint Tracks

9.4.4.1 Introduction

This Subclause specifies the reception hint track format for the secure real-time control protocol (SRTCP), defined in IETF RFC 3711.

SRTCP is used for real-time transport of control information for a SRTP session over the Internet Protocol. SRTCP takes for SRTP the role that RTCP takes for RTP, cf., 9.4.2. During streaming, each SRTP stream typically has an accompanying SRTCP stream that carries control information for the SRTP stream. One SRTCP reception hint track carries one SRTCP stream and is associated to the corresponding SRTP reception hint track through a track reference.

The format of the SRTCP reception hint tracks allows the storage of SRTCP Packets in the hint samples, e.g., of SRTCP Sender Reports.

The SRTCP Sender Reports are of particular interest for stream recording, because they reflect the current status of the server, e.g., the relationship of the media timing (SRTP timestamp of audio/video packets) to the server time (absolute time in NTP format). Knowledge of this relationship is also necessary for playback of recorded SRTP reception hint tracks in order to be able to detect and correct clock drift and jitter.

The timestamp synchrony box as specified in 9.4.1.2 makes it possible to correct clock drift and jitter before playing a file, and therefore recording of SRTCP streams is optional.

There is no server hint track equivalent for the SRCTP reception hint track, since SRTCP messages are generated on-the-fly during transmission.

9.4.4.2 General

There shall be zero or one SRTCP reception hint track for each SRTP reception hint track. An SRTCP reception hint track shall contain a track reference box including a reference of type '`cdsc`' to the associated SRTP reception hint track.

When i is the sample number a sample, the sample time DT(i) as specified in 8.6.1.2 indicates the reception time of the packet. The clock source for the reception time shall be the same as for the associated SRTP reception hint track. The value of `timescale` in the Media Header Box of an SRTCP reception hint track shall be equal to the value of `timescale` in the media header box of the associated SRTP reception hint track.

9.4.4.3 Sample Description Format

The entry-format in the sample description for the SRTCP reception hint tracks is 'stcp'. It is otherwise identical in structure to the sample entry format for RTCP. The encryption and authentication method of the SRTCP hint tracks are defined by the respective entries in SRTP Process box of the corresponding SRTP hint track.

NOTE: An equivalent to the ROC boxes defined for SRTP is not necessary for SRTCP, as the SRTCP packet contains an explicitly signalled initialization vector.

9.4.4.4 Sample Format

Sample format is the sample format for RTCP reception hint tracks as defined in 9.4.2.4.

9.4.5 Protected RTP Reception Hint Track

9.4.5.1 Introduction

This specification defines a mechanism for marking media streams as protected. This works by changing the four character code of the SampleEntry, and appending boxes containing both details of the protection mechanism and the original four character code. However, in this case the track is not protected; it is an 'in the clear' hint track which contains protected data. This Subclause describes the how reception hint tracks should be marked as carrying protected data, using a similar mechanism, and utilizing the same boxes.

9.4.5.2 Syntax

```
Class ProtectedRtpReceptionHintSampleEntry
   extends RtpReceptionHintSampleEntry ('prtp') {
   ProtectionSchemeInfoBox    SchemeInformation;
}
```

9.4.5.3 Semantics

The `SchemeInformation` ('sinf') box shall contain details of the protection scheme applied. This shall include the `OriginalFormatBox` which shall contain the four character code 'rrtp' (the four character code of the original RTPReceptionHintSampleEntry box).

9.4.6 Recording Procedure

See Annex H.

9.4.7 Parsing Procedure

See Annex H.

10 Sample Groups

10.1 Random Access Recovery Points

10.1.1.1 Definition

In some coding systems it is possible to random access into a stream and achieve correct decoding after having decoded a number of samples. This is known as gradual decoding refresh. For example, in video, the encoder might encode intra-coded macroblocks in the stream, such that it knows that within a certain period the entire picture consists of pixels that are only dependent on intra-coded macroblocks supplied during that period.

Samples for which such gradual refresh is possible are marked by being a member of this group. The definition of the group allows the marking to occur at either the beginning of the period or the end. However, when used with a particular media type, the usage of this group may be restricted to marking only one end (i.e. restricted to only positive or negative roll values). A roll-group is defined as that group of samples having the same roll distance.

10.1.1.2 Syntax

```
class VisualRollRecoveryEntry() extends VisualSampleGroupEntry ('roll')
{
   signed int(16) roll_distance;
}

class AudioRollRecoveryEntry() extends AudioSampleGroupEntry ('roll')
{
   signed int(16) roll_distance;
}
```

10.1.1.3 Semantics

`roll_distance` is a signed integer that gives the number of samples that must be decoded in order for a sample to be decoded correctly. A positive value indicates the number of samples after the sample that is a group member that must be decoded such that at the last of these recovery is complete, i.e. the last sample is correct. A negative value indicates the number of samples before the sample that is a group member that must be decoded in order for recovery to be complete at the marked sample. The value zero must not be used; the sync sample table documents random access points for which no recovery roll is needed.

10.2 Rate Share Groups

10.2.1 Introduction

Rate share instructions are used by players and streaming servers to help allocating bitrates dynamically when several streams share a common bandwidth resource. The instructions are stored in the file as sample group entries and apply when scalable or alternative media streams at different bitrates are combined with other scalable or alternative tracks. The instructions are time-dependent as samples in a track may be associated with different sample group entries. In the simplest case, only one target rate share value is specified per media and time range as illustrated in Figure 5.

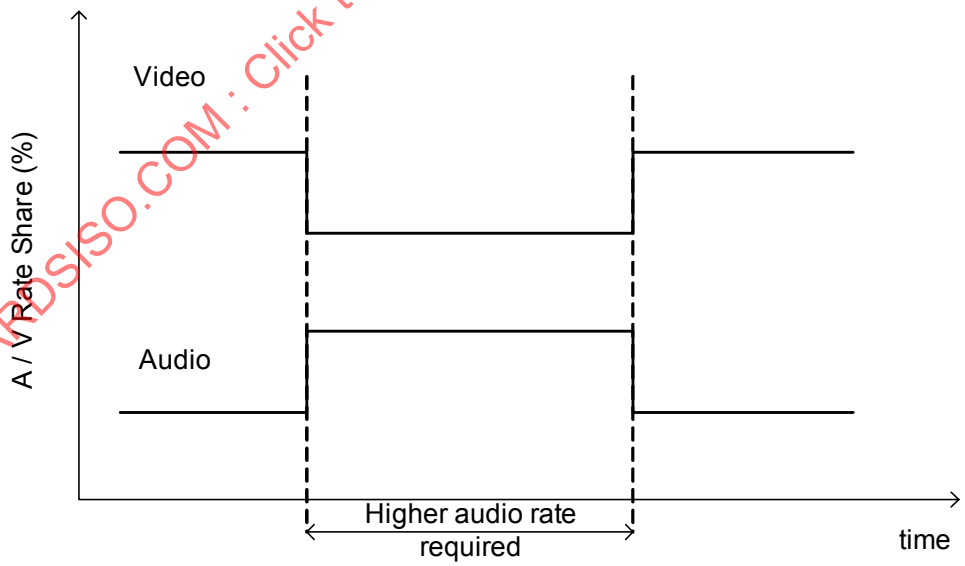

Figure 5 — Audio/Video rate share as function of time

In order to accommodate for rate share values that vary with the available bitrate, it is possible to specify more than one operation range. One may for instance indicate that audio requires a higher percentage (than video) at low available bitrates. Technically this is done by specifying two operation points as shown in Figure 6.

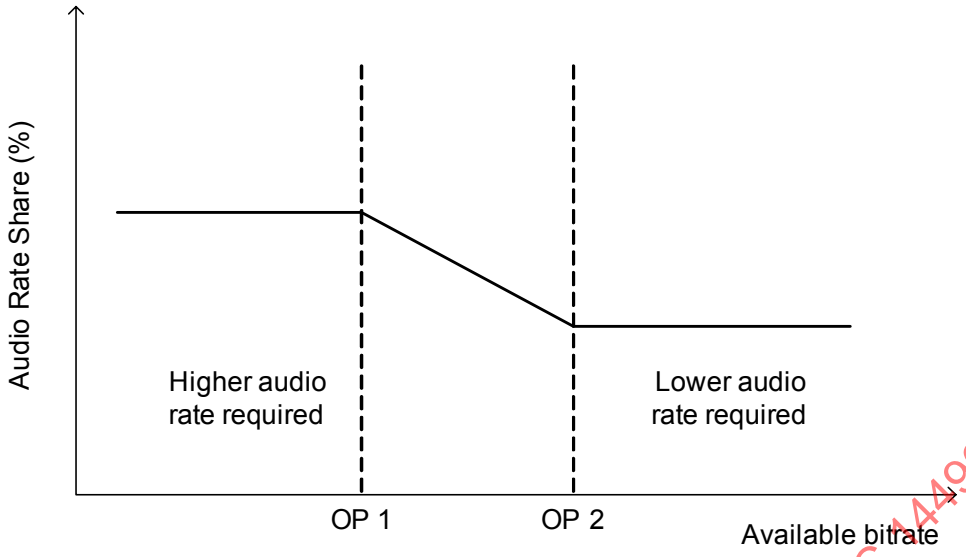

Figure 6 — Audio rate share as function of available bitrate

Operation points are defined in terms of total available bandwidth. For more complex situations it is possible to specify more operation points.

In addition to target rate share values, it is also possible to specify maximum and minimum bitrates for a certain media, as well as discard priority.

10.2.2 Rate Share Sample Group Entry

10.2.2.1 Definition

Each sample of a track may be associated to (zero or) one of a number of sample group descriptions, each of which defines a record of rate-share information. Typically the same rate-share information applies to many consecutive samples and it may therefore be enough to define two or three sample group descriptions that can be used at different time intervals.

The grouping type `'rash'` (short for rate share) is defined as the grouping criterion for rate share information. Zero or one sample-to-group box (`'sbgp'`) for the grouping type `'rash'` can be contained in the sample table box (`'stbl'`) of a track. It shall reside in a hint track, if a hint track is used, otherwise in a media track.

Target rate share may be specified for several operation points that are defined in terms of the total available bitrate, i.e., the bitrate that should be shared. If only one operation point is defined, the target rate share applies to all available bitrates. If several operation points are defined, then each operation point specifies a target rate share. Target rate share values specified for the first and the last operation points also specify the target rate share values at lower and higher available bitrates, respectively. The target rate share between two operation points is specified to be in the range between the target rate shares of those operation points. One possibility is to estimate with linear interpolation.

10.2.2.2 Syntax

```
class RateShareEntry() extends SampleGroupDescriptionEntry('rash') {
   unsigned int(16)  operation_point_count;
   if (operation_point_count == 1) {
      unsigned int(16)     target_rate_share;
   }
   else {
      for (i=0; i < operation_point_count; i++) {
         unsigned int(32)  available_bitrate;
         unsigned int(16)  target_rate_share;
      }
   }
   unsigned int(32)  maximum_bitrate;
   unsigned int(32)  minimum_bitrate;
   unsigned int(8)   discard_priority;
}
```

10.2.2.3 Semantics

`operation_point_count` is a non-zero integer that gives the number of operation points.

`available_bitrate` is a positive integer that defines an operation point (in kilobits per second). It is the total available bitrate that can be allocated in shares to tracks. Each entry shall be greater than the previous entry.

`target_rate_share` is an integer. A non-zero value indicates the percentage of available bandwidth that should be allocated to the media for each operation point. The value of the first (last) operation point applies to lower (higher) available bitrates than the operation point itself. The target rate share between operation points is bounded by the target rate shares of the corresponding operation points. A zero value indicates that no information on the preferred rate share percentage is provided.

`maximum_bitrate` is an integer. A nonzero value indicates (in kilobits per second) an upper threshold for which bandwidth should be allocated to the media. A higher bitrate than maximum bitrate should only be allocated if all other media in the session has fulfilled their quotas for target rate-share and maximum bitrate, respectively. A zero value indicates that no information on maximum bitrate is provided.

`minimum_bitrate` is an integer. A nonzero value indicates (in kilobits per second) a lower threshold for which bandwidth should be allocated to the media. If the allocated bandwidth would correspond to a smaller value, then no bitrate should be allocated. Instead preference should be given to other media in the session or alternate encodings of the same media. Zero minimum bitrate indicates that no information on minimum bitrate is provided.

`discard_priority` is an integer indicating the priority of the track when tracks are discarded to meet the constraints set by target rate share, maximum bitrate and minimum bitrate. Tracks are discarded in discard priority order and the track that has the highest discard priority value is discarded first.

10.2.3 Relationship between tracks

The purpose of defining rate share information is to aid a server or player extracting data from a track in combination with other tracks. Note that a server/player streams/plays tracks simultaneously if they belong to different alternate groups and can switch between tracks that belong to the same switch group within an alternate group. By default, all tracks are served/played simultaneously if no alternate groups are defined.

Rate share information should be provided for each track. A track that does not include rate share information has one operation point and can be treated as a constant-bitrate track with discard priority 128. Target rate share, minimum and maximum bitrates do not apply in this case.

Tracks that are alternates to each other shall (at each instance of time) define the same number of operation points at the same set of total available bitrates and have the same discard priorities. Note that the number and definition of operation points may depend on time. Alternate tracks may have different target rate shares, minimum and maximum bitrates.

10.2.4 Bitrate allocation

Rate share information on maximum bitrate, minimum bitrate, and target rate share can be combined for a track. If this is the case, the target rate share shall be applied to find an allocated bitrate before the impact of the maximum and minimum bitrates is considered.

When allocating bandwidth to several tracks, the following considerations apply:

1. In the case all tracks have explicit target rate share values and they don't sum up to 100 per cent, treat them as weights, i.e., normalize them.
2. The total allocation shall not exceed total available bitrate.
3. In a choice between alternate tracks, the chosen track should be the track that causes the alternate group to have an allocation most closely in accord with its target rate share, or the track that desires the highest bitrate that can be allocated without discarding other tracks (see below).
4. Tracks must have an allocation between their minimum and maximum bitrates, or be discarded.
5. Tracks should have an allocation in accord with their target rate shares, but this may be distorted to allow some tracks to achieve their minima, or in case some have reached their maxima.
6. If an allocation cannot be done including a track from every alternate group, then tracks should be discarded in discard priority order.
7. The allocation must be re-calculated whenever the operating set for an active track (one that has been selected from an alternate group) changes or the available bitrate changes.

10.3 Alternative Startup Sequences

10.3.1 Definition

An alternative startup sequence contains a subset of samples of a track within a certain period starting from a sync sample or a sample marked by 'rap ' sample grouping, which are collectively referred to as the initial sample below. By decoding this subset of samples, the rendering of the samples can be started earlier than in the case when all samples are decoded.

An 'alst' sample group description entry indicates the number of samples in any of the respective alternative startup sequences, after which all samples should be processed.

Either version 0 or version 1 of the Sample to Group Box may be used with the alternative startup sequence sample grouping. If version 1 of the Sample to Group Box is used, grouping_type_parameter has no defined semantics but the same algorithm to derive alternative startup sequences should be used consistently for a particular value of grouping_type_parameter.

A player utilizing alternative startup sequences could operate as follows. First, an initial sync sample from which to start decoding is identified by using the Sync Sample Box, the sample_is_non_sync_sample flag for samples enclosed in track fragments, or the 'rap ' sample grouping. Then, if the initial sync sample is associated to a sample group description entry of type 'alst' where roll_count is greater than 0, the player can use the alternative startup sequence. The player then decodes only those samples that are mapped to the alternative startup sequence until the number of samples that have been decoded is equal to roll_count. After that, all samples are decoded.

10.3.2 Syntax

```
class AlternativeStartupEntry() extends VisualSampleGroupEntry ('alst')
{
  unsigned int(16) roll_count;
  unsigned int(16) first_output_sample;
  for (i=1; i <= roll_count; i++)
    unsigned int(32) sample_offset[i];
  j=1;
  do { // optional, until the end of the structure
    unsigned int(16) num_output_samples[j];
    unsigned int(16) num_total_samples[j];
    j++;
  }
}
```

10.3.3 Semantics

`roll_count` indicates the number of samples in the alternative startup sequence. If `roll_count` is equal to 0, the associated sample does not belong to any alternative startup sequence and the semantics of `first_output_sample` are unspecified. The number of samples mapped to this sample group entry per one alternative startup sequence shall be equal to `roll_count`.

`first_output_sample` indicates the index of the first sample intended for output among the samples in the alternative startup sequence. The index of the sync initial sample starting the alternative startup sequence is 1, and the index is incremented by 1, in decoding order, per each sample in the alternative startup sequence.

`sample_offset[i]` indicates the decoding time delta of the i-th sample in the alternative startup sequence relative to the regular decoding time of the sample derived from the Decoding Time to Sample Box or the Track Fragment Header Box. The sync initial sample starting the alternative startup sequence is its first sample.

`num_output_samples[j]` and `num_total_samples[j]` indicate the sample output rate within the alternative startup sequence. The alternative startup sequence is divided into k consecutive pieces, where each piece has a constant sample output rate which is unequal to that of the adjacent pieces. The first piece starts from the sample indicated by `first_output_sample`. `num_output_samples[j]` indicates the number of the output samples of the j-th piece of the alternative startup sequence. `num_total_samples[j]` indicates the total number of samples, including those that are not in the alternative startup sequence, from the first sample in the j-th piece that is output to the earlier one (in composition order) of the sample that ends the alternative startup sequence and the sample that immediately precedes the first output sample of the (j+1)th piece.

10.3.4 Examples

Hierarchical temporal scalability (e.g., in AVC and SVC) improves compression efficiency but increases the decoding delay due to reordering of the decoded pictures from the (de)coding order to output order. Deep temporal hierarchies have been demonstrated to useful in terms of compression efficiency in some studies. When the temporal hierarchy is deep and the operation speed of the decoder is limited (to no faster than real-time processing), the initial delay from the start of the decoding to the start of rendering is substantial and may affect the end-user experience negatively.

Figure 7 illustrates a typical hierarchically scalable bitstream with five temporal levels. Figure 7a shows the example sequence in output order. Values enclosed in boxes indicate the frame_num value of the picture. Values in italics indicate a non-reference picture while the other pictures are reference pictures. Figure 7b shows the example sequence in decoding order. Figure 7c shows the example sequence in output order when assuming that the output timeline coincides with that of the decoding timeline and the decoding of one picture lasts one picture interval. It can be seen that playback of the stream starts five picture intervals later than the decoding of the stream started. If the pictures were sampled at 25 Hz, the picture interval is 40 msec, and the playback is delayed by 0.2 sec.

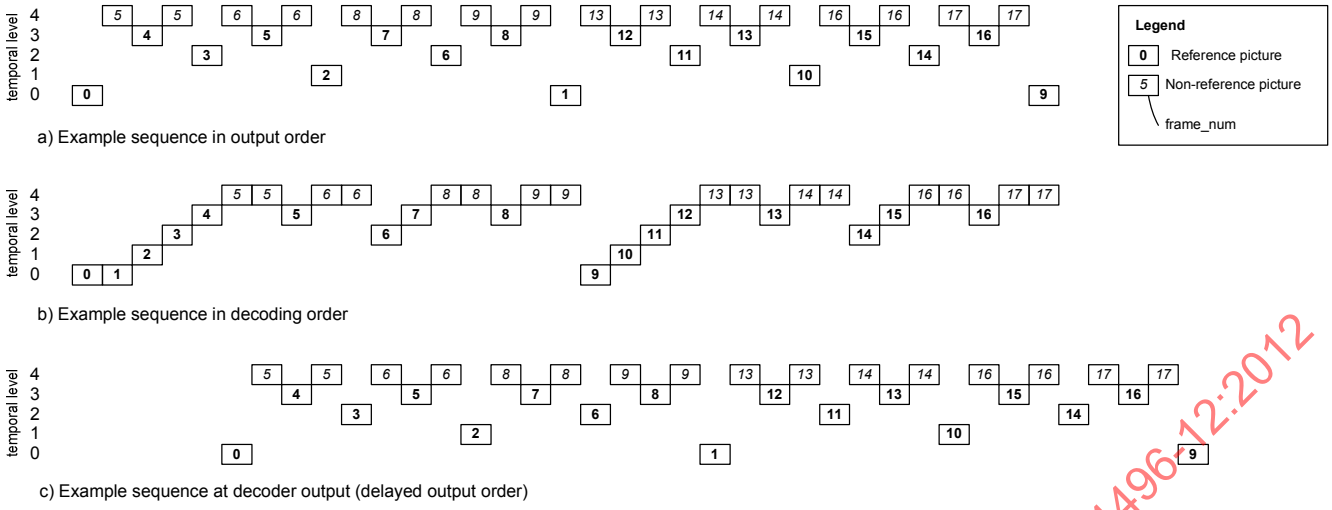

Figure 7 — Decoded picture buffering delay of an example sequence with five temporal levels

Thanks to the temporal hierarchy, it is possible to decode only a subset of the pictures at the beginning of the sequence. Consequently, rendering can be started faster but the displayed picture rate is lower at the beginning. In other words, a player can make a trade-off between the duration of the initial startup delay and the initial displayed picture rate. Figure 8 and Figure 9 show two examples of alternative startup sequences where a subset of the bitstream of Figure 7 is decoded.

The samples selected for decoding and the decoder output are presented in Figure 8a and Figure 8b, respectively. The reference picture having frame_num equal to 4 and the non-reference pictures having frame_num equal to 5 are not decoded. In this example, the rendering of pictures starts four picture intervals earlier than in Figure 7. When the picture rate is 25 Hz, the saving in startup delay is 160 msec. The saving in the startup delay comes with the disadvantage of a lower displayed picture rate at the beginning of the bitstream.

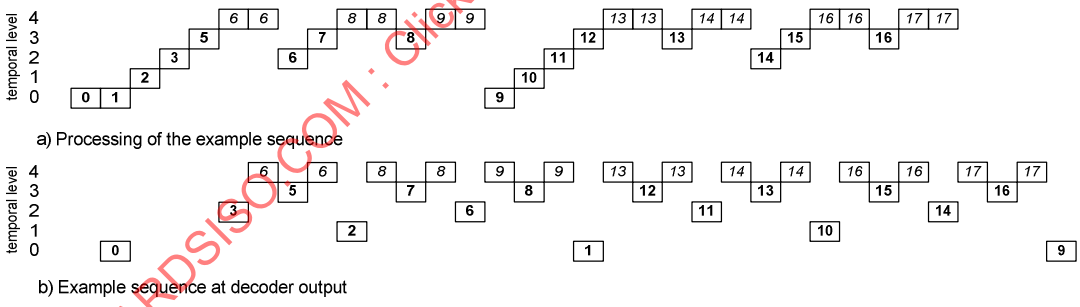

Figure 8 — An example of an alternative startup sequence

In the example of Figure 9, another way of selecting the pictures for decoding is presented. The decoding of the pictures that depend on the picture with frame_num equal to 3 is omitted and the decoding of non-reference pictures within the second half of the first group of pictures is omitted too. The decoded picture resulting from the sample with frame_num equal to 2 is the first one that is output. As a result, the output picture rate of the first group of pictures is half of normal picture rate, but the display process starts two frame intervals (80 msec in 25 Hz picture rate) earlier than in the conventional solution illustrated in Figure 7.

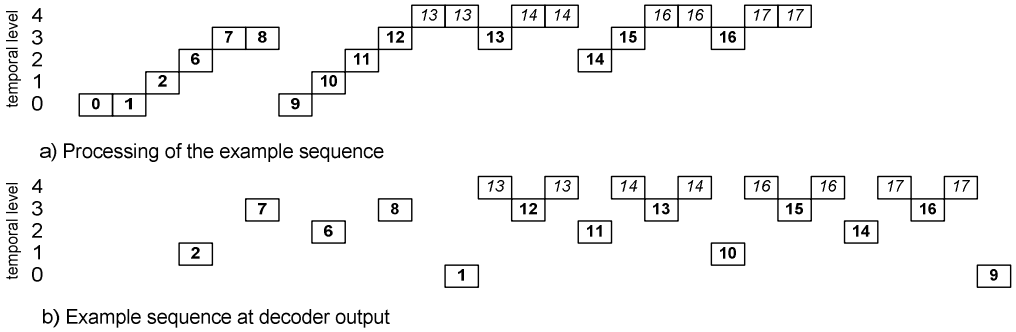

Figure 9 — Another example of an alternative startup sequence

10.4 Random Access Point (RAP) Sample Grouping

10.4.1 Definition

A sync sample is specified to be a random access point after which all samples in decoding order can be correctly decoded. However, it may be possible to encode an "open" random access point, after which all samples in output order can be correctly decoded, but some samples following the random access point in decoding order and preceding the random access point in output order need not be correctly decodable. For example, an intra picture starting an open group of pictures can be followed in decoding order by (bi-)predicted pictures that however precede the intra picture in output order; though they possibly cannot be correctly decoded if the decoding starts from the intra picture, they are not needed.

Such "open" random-access samples can be marked by being a member of this group. Samples marked by this group must be random access points, and may also be sync points (i.e. it is not required that samples marked by the sync sample table be excluded).

10.4.2 Syntax

```
class VisualRandomAccessEntry() extends VisualSampleGroupEntry ('rap ')
{
   unsigned int(1) num_leading_samples_known;
   unsigned int(7) num_leading_samples;
}
```

10.4.3 Semantics

`num_leading_samples_known` equal to 1 indicates that the number of leading samples is known for each sample in this group, and the number is specified by num_leading_samples. A leading sample is such a sample associated with an "open" random access point (RAP). It precedes the RAP in presentation order and immediate follows the RAP or another leading sample in decoding order, and when decoding starts from the RAP, the sample cannot be correctly decoded.

`num_leading_samples` specifies the number of leading samples for each sample in this group. When num_leading_samples_known is equal to 0, this field should be ignored.

10.5 Temporal level sample grouping

10.5.1 Definition

Many video codecs support temporal scalability where it is possible to extract one or more subsets of frames that can be independently decoded. A simple case is the extraction of I frames for a bitstream with a regular I-frame interval, e.g,, IPPPIPPP…, where every 4th picture is an I frame. Also subsets of these I frames can be extracted for even lower frame rates. More elaborate situations with several temporal levels can be constructed using hierarchical B or P frames.

The Temporal Level sample grouping ('tele') provides a codec-independent sample grouping that can be used to group samples (access units) in a track (and potential track fragments) according to temporal level, where samples of one temporal level have no coding dependencies on samples of higher temporal levels. The temporal level equals the sample group description index (taking values 1, 2, 3, etc). The bitstream containing only the access units from the first temporal level to a higher temporal level remains conforming to the coding standard.

A grouping according to temporal level facilitates easy extraction of temporal subsequences, for instance using the Subsegment Indexing box in 8.16.4.

10.5.2 Syntax

```
class TemporalLevelEntry() extends VisualSampleGroupEntry('tele')
{
   bit(1)   level_independently_decodable;
   bit(7)   reserved=0;
}
```

10.5.3 Semantics

The temporal level of samples in a sample group equals to the sample group description index.

`level_independently_decodable` is a flag. 1 indicates that all samples of this level have no coding dependencies on samples of other levels. 0 indicates that no information is provided.

11 Extensibility

11.1 Objects

The normative objects defined in this specification are identified by a 32-bit value, which is normally a set of four printable characters from the ISO 8859-1 character set.

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

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

User objects use the escape type `'uuid'`. They are documented above in subclause 6.2. After the size and type fields, there is a full 16-byte UUID.

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

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

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

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

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

11.2 Storage formats

The main file containing the metadata may use other files to contain media-data. These other files may contain header declarations from a variety of standards, including this one.

If such a secondary file has a metadata declaration set in it, that metadata is not part of the overall presentation. This allows small presentation files to be aggregated into a larger overall presentation by building new metadata and referencing the media-data, rather than copying it.

The references into these other files need not use all the data in those files; in this way, a subset of the media-data may be used, or unwanted headers ignored.

11.3 Derived File formats

This specification may be used as the basis as the specific file format for a restricted purpose: for example, the MP4 file format for MPEG-4 and the Motion JPEG 2000 file format are both derived from it. When a derived specification is written, the following must be specified:

The name of the new format, and its brand and compatibility types for the File Type Box. Generally a new file extension will be used, a new MIME type, and Macintosh file type also, though the definition and registration of these are outside the scope of this specification.

Any template fields used must be explicitly declared; their use must be conformant with the specification here.

The exact 'codingname' and 'protocol' identifiers as used in the Sample Description must be defined. The format of the samples that these code-points identify must also be defined. However, it may be preferable to fit the new coding systems into an existing framework (e.g. the MPEG-4 systems framework), than to define new coding points at this level. For example, a new audio format could use a new codingname, or could use 'mp4a' and register new identifiers within the MPEG-4 audio framework.

New boxes may be defined, though this is discouraged.

If the derived specification needs a new track type other than visual or audio, then a new handler-type must be registered. The media header required for this track must be identified. If it is a new box, it must be defined and its box type registered. In general, it is expected that most systems can use existing track types.

Any new track reference types should be registered and defined.

As defined above, the Sample Description format may be extended with optional or required boxes. The usual syntax for doing this would be to define a new box with a specific name, extending (for example) Visual Sample Entry, and containing new boxes.

Annex A
(informative)

Overview and Introduction

A.1 Section Overview

This section provides an introduction to the file format, that potentially assists readers in understanding the overall concepts underlying the file format. It forms an informative annex to this specification.

A.2 Core Concepts

In the file format, the overall presentation is called a **movie**. It is logically divided into **track**s; each track represents a timed sequence of media (frames of video, for example). Within each track, each timed unit is called a **sample**; this might be a frame of video or audio. Samples are implicitly numbered in sequence. Note that a frame of audio may decompress into a sequence of audio samples (in the sense this word is used in audio); in general, this specification uses the word sample to mean a timed frame or unit of data. Each track has one or more **sample description**s; each sample in the track is tied to a description by reference. The description defines how the sample may be decoded (e.g. it identifies the compression algorithm used).

Unlike many other multi-media file formats, this format, with its ancestors, separates several concepts that are often linked. Understanding this separation is key to understanding the file format. In particular:

The physical structure of the file is not tied to the physical structures of the media itself. For example, many file formats 'frame' the media data, putting headers or other data immediately before or after each frame of video; this file format does not do this.

Neither the physical structure of the file, nor the layout of the media, is tied to the time ordering of the media. Frames of video need not be laid down in the file in time order (though they may be).

This means that there are file structures that describe the placement and timing of the media; these file structures permit, but do not require, time-ordered files.

All the data within a conforming file is encapsulated in **box**es (called **atoms** in predecessors of this file format). There is no data outside the box structure. All the metadata, including that defining the placement and timing of the media, is contained in structured boxes. This specification defines the boxes. The media data (frames of video, for example) is referred to by this metadata. The media data may be in the same file (contained in one or more boxes), or can be in other files; the metadata permits referring to other files by means of URLs. The placement of the media data within these secondary files is entirely described by the metadata in the primary file. They need not be formatted to this specification, though they may be; it is possible that there are no boxes, for example, in these secondary media files.

Tracks can be of various kinds. Three are important here. **Video track**s contain samples that are visual; **audio track**s contain audio media. **Hint track**s are rather different; they contain instructions for a streaming server in how to form packets for a streaming protocol, from the media tracks in a file. Hint tracks can be ignored when a file is read for local playback; they are only relevant to streaming.

A.3 Physical structure of the media

The boxes that define the layout of the media data are found in the sample table. These include the data reference, the sample size table, the sample to chunk table, and the chunk offset table. Between them, these tables allow each sample in a track to be both located, and its size to be known.

The **data reference**s permit locating media within secondary media files. This allows a composition to be built from a 'library' of media in separate files, without actually copying the media into a single file. This greatly facilitates editing, for example.

The tables are compacted to save space. In addition, it is expected that the interleave will not be sample by sample, but that several samples for a single track will occur together, then a set of samples for another track, and so on. These sets of contiguous samples for one track are called **chunk**s. Each chunk has an offset into its containing file (from the beginning of the file). Within the chunk, the samples are contiguously stored. Therefore, if a chunk contains two samples, the position of the second may be found by adding the size of the first to the offset for the chunk. The chunk offset table provides the offsets; the sample to chunk table provides the mapping from sample number to chunk number.

Note that in between the chunks (but not within them) there may be 'dead space', un-referenced by the media data. Thus, during editing, if some media data is not needed, it can simply be left unreferenced; the data need not be copied to remove it. Likewise, if the media data is in a secondary file formatted to a 'foreign' file format, headers or other structures imposed by that foreign format can simply be skipped.

A.4 Temporal structure of the media

Timing in the file can be understood by means of a number of structures. The movie, and each track, has a **timescale**. This defines a time axis which has a number of ticks per second. By suitable choice of this number, exact timing can be achieved. Typically, this is the sampling rate of the audio, for an audio track. For video, a suitable scale should be chosen. For example, a media `TimeScale` of 30000 and media sample durations of 1001 exactly define NTSC video (often, but incorrectly, referred to as 29.97) and provide 19.9 hours of time in 32 bits.

The time structure of a track may be affected by an **edit list**. These provide two key capabilities: the movement (and possible re-use) of portions of the time-line of a track, in the overall movie, and also the insertion of 'blank' time, known as empty edits. Note in particular that if a track does not start at the beginning of a presentation, an initial empty edit is needed.

The overall duration of each track is defined in headers; this provides a useful summary of the track. Each sample has a defined **duration**. The exact presentation time (its time-stamp) of a sample is defined by summing the durations of the preceding samples.

A.5 Interleave

The temporal and physical structures of the file may be aligned. This means that the media data has its physical order within its container in time order, as used. In addition, if the media data for multiple tracks is contained in the same file, this media data would be interleaved. Typically, in order to simplify the reading of the media data for one track, and to keep the tables compact, this interleave is done at a suitable time interval (e.g. 1 second), rather than sample by sample. This keeps the number of chunks down, and thus the chunk offset table small.

A.6 Composition

If multiple audio tracks are contained in the same file, they are implicitly mixed for playback. This mixing is affected by the overall track **volume**, and the left/right **balance**.

Likewise, video tracks are composed, by following their layer number (from back to front), and their composition mode. In addition, each track may be transformed by means of a matrix, and also the overall movie transformed by **matrix**. This permits both simple operations (e.g. pixel doubling, correction of 90º rotation) as well as more complex operations (shearing, arbitrary rotation, for example).

Derived specifications may over-ride this default composition of audio and video with more powerful systems (e.g. MPEG-4 BIFS).

A.7 Random access

This section describes how to seek. Seeking is accomplished primarily by using the child boxes contained in the sample table box. If an edit list is present, it must also be consulted.

If you want to seek a given track to a time T, where T is in the time scale of the movie header box, you could perform the following operations:

1) If the track contains an edit list, determine which edit contains the time T by iterating over the edits. The start time of the edit in the movie time scale must then be subtracted from the time T to generate T', the duration into the edit in the movie time scale. T' is next converted to the time scale of the track's media to generate T''. Finally, the time in the media scale to use is calculated by adding the media start time of the edit to T''.

2) The time-to-sample box for a track indicates what times are associated with which sample for that track. Use this box to find the first sample prior to the given time.

3) The sample that was located in step 1 may not be a sync sample. The sync sample table indicates which samples are in fact random access points. Using this table, you can locate which is the first sync sample prior to the specified time. The absence of the sync sample table indicates that all samples are synchronization points, and makes this problem easy. Having consulted the sync sample table, you probably wish to seek to whichever resultant sample is closest to, but prior to, the sample found in step 1.

4) At this point you know the sample that will be used for random access. Use the sample-to-chunk table to determine in which chunk this sample is located.

5) Knowing which chunk contained the sample in question, use the chunk offset box to figure out where that chunk begins.

6) Starting from this offset, you can use the information contained in the sample-to-chunk box and the sample size box to figure out where within this chunk the sample in question is located. This is the desired information.

A.8 Fragmented movie files

This section introduces a technique that may be used in ISO files, where the construction of a single Movie Box in a movie is burdensome. This can arise in at least the following cases:

- Recording. At the moment, if a recording application crashes, runs out of disk, or some other incident happens, after it has written a lot of media to disk but before it writes the Movie Box, the recorded data is unusable. This occurs because the file format insists that all metadata (the Movie Box) be written in one contiguous area of the file.

- Recording. On embedded devices, particularly still cameras, there is not the RAM to buffer a Movie Box for the size of the storage available, and re-computing it when the movie is closed is too slow. The same risk of crashing applies, as well.

- HTTP fast-start. If the movie is of reasonable size (in terms of the Movie Box, if not time), the Movie Box can take an uncomfortable period to download before fast-start happens.

The basic 'shape' of the movie is set in initial Movie Box: the number of tracks, the available sample descriptions, width, height, composition, and so on. However the Movie Box does not contain the information for the full duration of the movie; in particular, it may have few or no samples in its tracks.

To this minimal or empty movie, extra samples are added, in structure called movie fragments.

The basic design philosophy is the same as in the Movie Box; data is not 'framed'. However, the design is such that it can be treated as a 'framing' design if that is needed. The structures map readily to the Movie Box, so an fragmented presentation can be rewritten as a single Movie Box.

The approach is that defaults are set for each sample, both globally (once per track) and within each fragment. Only those fragments that have non-default values need include those values. This makes the common case — regular, repeating, structures — compact, without disabling the incremental building of movies that have variations.

The regular Movie Box sets up the structure of the movie. It may occur anywhere in the file, though it is best for readers if it precedes the fragments. (This is not a rule, as trivial changes to the Movie Box that force it to the end of the file would then be impossible). This Movie Box:

- must represent a valid movie in its own right (though the tracks may have no samples at all);
- has an box in it to indicate that fragments should be found and used;
- is used to contain the complete edit list (if any).

Note that software that doesn't understand fragments will play just this initial movie. Software that does understand fragments and gets a non-fragmented movie won't scan for fragments as the fragment indication box won't be found.

Annex B
(informative)

Patent Statements

The International Organization for Standardization and the International Electrotechnical Commission (IEC) draw attention to the fact that it is claimed that compliance with this part of ISO/IEC 14496 and ISO/IEC 15444 may involve the use of patents.

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

The holders of these patent rights have assured the ISO and IEC that they are willing to negotiate licenses under reasonable and non-discriminatory terms and conditions with applicants throughout the world. In this respect, the statements of the holders of these patents right are registered with ISO and IEC. Information may be obtained from the companies listed below.

Attention is drawn to the possibility that some of the elements of this part of ISO/IEC 14496 and ISO/IEC 15444 may be the subject of patent rights other than those identified in this annex. 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.

Please note that Patent statements that apply to the ISO Base Media File Format may not apply to an implementation of ISO/IEC 15444-3 (Motion JPEG 2000). ISO/IEC 15444-3 uses a subset of ISO/IEC 15444-12 (The ISO Base Media File Format).

	Company
1.	Apple, Inc.
2.	Electronics & Telecommunications Research Institute
3.	Matsushita Electric Industrial Co., Ltd
4.	Nokia Corporation
5.	Nokia Mobile Phones Ltd
6.	QUALCOMM Incorporated
7.	Technical Research Centre of Finland
8.	Telefonaktiebolaget LM Ericsson

Annex C
(informative)

Guidelines on deriving from this specification

C.1 Introduction

This Annex provides informative text to explain how to derive a specific file format from the ISO Base Media File Format.

ISO/IEC 14496-12 | ISO/IEC 15444-12 ISO Base Media Format defines the basic structure of the file format. Media-specific and user-defined extensions can be provided in other specifications that are derived from the ISO Base Media File Format.

C.2 General Principles

C.2.1 General

A number of existing file formats use the ISO Base Media File Format, not least the MPEG-4 MP4 File Format (ISO/IEC 14496-14), and the Motion JPEG 2000 MJ2 File Format (ISO/IEC 15444-3). When considering a new specification derived from the ISO Base Media File format, all the existing specifications should be used both as examples and a source of definitions and technology. Check with the registration authority to find what might already exist, and what specifications exist.

In particular, if an existing specification already covers how a particular media type is stored in the file format (e.g. MPEG-4 video in MP4), that definition should be used and a new one should not be invented. In this way specifications which share technology will also share the definition of how that technology is represented.

Be as permissive as possible with respect to the presence of other information in the file; indicate that unrecognized boxes and media may be ignored (not “should be ignored”). This permits the creation of hybrid files, drawing from more than one specification, and the creation of multi-format players, capable of handling more than one specification.

When layering on this specification, it's worth observing that there are some characteristics that are intentionally ‘parameters’ to the lower (Part 12) specification, that need to be specified. Equally, there are some characteristics of the Part 12 file format specification that are internal and should rarely be discussed by other specifications. Of course, there are some characteristics in a grey area in between.

Derived specifications are ideally written solely in terms of the parameters of the Part 12 file format; what a sample is, what its timestamps mean, and so on. Mentioning specific existing boxes in a derived specification may often turn out to be an error, except in limited cases (e.g. adding a user-data box, or an extension box).

C.2.2 Base layer operations

It should be possible to perform some operations on a Part 12 file without knowing anything about any potential derived specifications. These operations might include the obvious reading tracks, finding the data and timing for samples, and their sample description and track type, and so on. This might be done, for example, by a file-format inspector or general library like the reference software.

Less obvious are a class of manipulations of the files:

a) re-interleaving the data; making the media data in time order, with the samples for various tracks grouped into chunks of a sensible size, with the chunks interleaved;

b) making files that use data references self-contained, by copying the data from external files into the new file;
c) removing free space atoms and compacting the atom structure;
d) removing data from 'mdat' atoms that appears to be un-referenced by tracks or meta-data atoms;
e) removing sample entries that have no associated samples;
f) removing sample groups that have no associated samples;
g) extracting some tracks and making a new file with just those (e.g. an audio track from an audio/video presentation);
h) inserting, or removing, movie fragments, or re-fragmenting a movie.

This list is not exhaustive, of course.

C.3 Boxes

You can add boxes to the file format, but be careful about how they interact with other boxes. In particular, if they 'cross-link' into existing boxes, you might not be able to mark such files as compliant with Part 12.

You must register all new boxes, except those using the 'uuid' type. Likewise, you should register codec (sample entry) names, brands, track reference types, handlers (media types), group types, and protection scheme types. It really is a bad idea to use one of these without registration, as collisions may occur – or someone else may register the same identifier with a different meaning.

You should not write a box using the 'UUID escape' (the reserved ISO UUID pattern 0xXXXXXXXX-0011-0010-8000-00AA00389B71, where the four-character code replaces the X's) if a simple four-character-code can be used, and ideally you shouldn't design to use a UUID box: it's better to place your data in known 'expansion points' of the file format if at all possible, or register a new box type if really needed.

Don't forget that *all* data in ISO files must be, or be contained in, boxes. You can introduce a signature, but it must 'look like' a box.

Do not require that any existing or new boxes you define be in a particular position, if at all possible. For example, the existing JPEG 2000 specifications require a signature box and that it be first in the file. If another specification also defines a signature box and also requires that it be first, then a file conformant to both specifications cannot be constructed.

It must be possible to 'walk' the top-level of a file by finding box lengths. Don't forget that 'implied length' is permitted at file level.

Unless absolutely unavoidable, boxes should contain either data (e.g. in fields), or other boxes, but not both. All boxes containing data should be a full box to allow later changes to syntax and semantics. Boxes containing other boxes are known as container boxes, and are normally a plain (non-full) box, since their semantics will never change if they are documented to contain only boxes.

C.4 Brand Identifiers

C.4.1 Introduction

This section covers the use of brand identifiers in the file-type box, including:
- Introduction of a new brand.
- Player's behaviour depending on the brand.
- Setting of the brand on the creation of the ISO Base Media file.

Brands identify a specification and make a simple set of statements:
a) the file conforms to all requirements of the identified specification;
b) the file contains nothing contrary to the identified specification;
c) a reader implementing potentially that single specification may read, interpret, and possibly present the file, ignoring data it does not recognize.

Specifications should therefore say (if they need a brand) “the brand that identifies files conformant to this specification is XXXX”, and register the brand.

C.4.2 Usage of the Brand

In order to identify the specifications to which the file complies, brands are used as identifiers in the file format. These brands are set win the File Type Box.

For example, a brand might indicate:

(1) the codecs that may be present in the file,
(2) how the data of each codec is stored,
(3) constraints and extensions that are applied to the file.

New brands may be registered if it is necessary to make a new specification that is not fully conformant to the existing standards. For example, 3GPP allows using AMR and H.263 in the file format. Since these codecs were not supported in any standards at that time, 3GPP specified the usage of the SampleEntry and template fields in the ISO Base Media Format as well as defining new boxes to which these codecs refer. Considering that the file format is used more widely in the future, it is expected that more brands will be needed.

Brands are not additive; they stand alone. You cannot say: “this brand indicates that support for Y is also required” because the ‘also’ has no referent.

Systems that re-write files should remove brands that they do not recognize, as they do not know whether the file still conforms to that brand’s requirements (e.g. re-interleaving a file may take it out of conformance with a specification that requires a certain style of interleaving).

Note that the major brand usually implies the file extension, which in turn implies the MIME type. But these are not rules. In addition, when serving under a MIME type do not forget that MIME types can take parameters, and the list of compatible brands would often be useful to the receiving system.

C.4.3 Introduction of a new brand

A new brand can be defined if conformance to a new specification must be indicated. This generally means that for the definition of a new brand at least one of the following conditions should be satisfied:

1. Use of a codec that is not supported in any existing brands.

2. Use more than one codec in a combination that is not supported in any existing brands. In addition, the playback of the file is allowed only when decoding of all the media in the file is supported by the player.

3. Use constraints and/or extensions (Boxes, template fields, etc.) that are user-specific.

However, the file format contains both a major_brand field and a compatible_brands array. These fields are owned by the file author and the part 12 specification. Do not write a specification that talks about these fields, merely about brands and what they mean. In particular, do *not* claim the major_brand field (“files conformant to this specification must set the major_brand to XXXX”) as a file could never be conformant to two such specifications written that way, and you also block someone also from deriving a specification from yours. However, brands that are only permitted as compatible brands may be defined.

Brands can be used as a tracer, however. It’s perfectly legal to have a brand which has no requirements, and is placed in a file as an ‘I was there’ point (or strictly “this brand requires that the file was last written by ZZZZ”).

C.4.4 Player Guideline

If more than one brand is present in the list of the compatible_brands, and one or more brands are supported by the player, the player shall play those aspects of the file that comply with those specifications. In this case, the player may not be able to decode unsupported media.

C.4.5 Authoring Guideline

If the author wants to create a file that complies with more than one specification, the following considerations apply:

1. There must be nothing contrary to the specification identified by a brand within the file. For example, if a specification requires that files be self-contained, then the brand indication of that specification must not be used on non-self-contained files.

2. If the author is satisfied that a player compliant with only one of the specifications play only that media compliant with that specification, then that brand may be indicated.

3. If the author requires that the media from more than one specification be played, then a new brand would be needed as this represents a new conformance requirement for the player.

C.4.6 Example

In this section, we take the example case when a new brand can be defined.

First of all, we explain about the two currently existing brands. If the brand '3gp5' is in the list of the compatible_brands, it indicates that the file contains the media defined in 3GPP TS 26.234 (Release 5) in the way specified by the standard. For example, the file of '3gp5' brand may contain H.263. Likewise, if the brand 'mp42' is in the list of the compatible_brands, it indicate that the file contains the media defined in the ISO/IEC 14496-14 in the specified way. For example, the file of 'mp42' brand may contain MP3. However, MP3 is not supported in '3gp5' brand.

Given that the file contains H.263 and MP3, and has '3gp5' and 'mp42' as the compatible_brands. If the player complies only with '3gp5' and does not support MP3, recommended behaviour of the player is to play only H.263. If the content's author does not expect such behaviour, a new brand is defined to indicate that both H.263 and MP3 are supported in the file. By specifying the newly defined brand in the list of the compatible_brands, it can prevent the above behaviour and the file is played only when the player supports both H.263 and MP3.

C.5 Storage of new media types

There are two choices in the definition of how a new media type should be stored.

First, if MPEG-4 systems constructs are desired or acceptable, then:

a) a new ObjectTypeIndication should be requested and used;
b) the decoderspecificinformation for this codec should be defined as an MPEG-4 descriptor;
c) the access unit format should be defined for this media.

The media then uses the MPEG-4 code-points in the file format; for example, a new video codec would use a sampleentry of type 'mp4v'.

If the MPEG-4 systems layer is not suitable or otherwise not desired, then:

a) a new sampleentry four-character code should be requested and used;
b) any additional information needed by the decoder should be defined as boxes to be stored within the sampleentry;
c) the file-format sample format should be defined for this media.

Note that in the second case, the registration authority will also allocate an objecttypeindication for use in MPEG-4 systems.