INTERNATIONAL STANDARD

ISO/IEC 14496-12

Second edition
2005-04-01

Corrected version
2005-10-01

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 14496-12:2005(E)

PDF disclaimer

This PDF file may contain embedded typefaces. In accordance with Adobe's licensing policy, this file may be printed or viewed but shall not be edited unless the typefaces which are embedded are licensed to and installed on the computer performing the editing. In downloading this file, parties accept therein the responsibility of not infringing Adobe's licensing policy. The ISO Central Secretariat accepts no liability in this area.

Adobe is a trademark of Adobe Systems Incorporated.

Details of the software products used to create this PDF file can be found in the General Info relative to the file; the PDF-creation parameters were optimized for printing. Every care has been taken to ensure that the file is suitable for use by ISO member bodies. In the unlikely event that a problem relating to it is found, please inform the Central Secretariat at the address given below.

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.

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

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 second edition cancels and replaces the first edition (ISO/IEC 14496-12:2004) which has been technically revised.

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*

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

— *Part 9: Reference hardware description*

— *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)*

The following parts are under preparation:

— *Part 21: MPEG-J extensions for rendering*

This corrected version of ISO/IEC 14496-12:2005 (E) incorporates the following correction:

— In 8.16.2, 8.17.2.1, 8.17.3.1, 8.18.2 and 8.19.2, "•" has been replaced with "⩽".

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 15444-12 and 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)".

Information technology — Coding of audio-visual objects —

Part 12:
ISO base media file format

1 Scope

This International Standard 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 referenced documents are indispensable for the application of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

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

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

ISO/IEC 14496-1:2004: *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*

ITU-T Rec.T.800 | 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*

IETF RFC 3711, *"The Secure Real-time Transport Protocol"*, Baugher M. et al., March 2004.

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

3 Definitions

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

3.1
Box:

An object-oriented building block defined by a unique type identifier and length (called 'atom' in some specifications, including the first definition of MP4).

3.2
Chunk:

A contiguous set of samples for one track.

3.3
Container Box:

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

3.4
Hint Track:

A special track which does not contain media data. Instead it contains instructions for packaging one or more tracks into a streaming channel.

3.5
Hinter:

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

3.6
Movie Box:

A container box whose sub-boxes define the metadata for a presentation ('moov').

3.7
Media Data Box:

A container box which can hold the actual media data for a presentation ('mdat').

3.8
ISO Base Media File:

The name of the file format described in this specification.

3.9
Presentation:

One or more motion sequences (q.v.), possibly combined with audio.

3.10
Sample:

In non-hint tracks, a sample is an individual frame of video, a time-contiguous series of video frames, or a time-contiguous compressed section of audio. In hint tracks, a sample defines the formation of one or more streaming packets. No two samples within a track may share the same time-stamp.

3.11
Sample Description:

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

3.12
Sample Table:

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

3.13
Track:

A collection of related samples (q.v.) in an ISO base media file. For media data, a track corresponds to a sequence of images or sampled audio. For hint tracks, a track corresponds to a streaming channel.

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 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.

```
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**

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 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.

The type 'isom' (ISO Base Media file) is defined in this section of this specification, as identifying files that conform to the first version of ISO Base Media File Format.

More specific identifiers can be used to identify precise versions of specifications providing more detail. This brand should not be used as the major brand; this base file format should be derived into another specification to be used. There is therefore no defined normal file extension, or mime type assigned to this brand, nor definition of the minor version when 'isom' is the major brand.

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.

The brand 'iso2' shall be used to indicate compatibility with this amended version of the ISO Base Media File Format; it may be used in addition to or instead of the 'isom' brand and the same usage rules apply. If used without the brand 'isom' identifying the first version of this specification, it indicates that support for some or all

of the technology introduced by this amendment is required, such as the functionality in sub-clauses [8.40] through [8.45], or the SRTP support in sub-clause [10], is required.

The brand 'avc1' shall be used to indicate that the file is conformant with the 'AVC Extensions' in sub-clause [8.40]. If used without other brands, this implies that support for those extensions is required. The use of 'avc1' as a major-brand may be permitted by specifications; in that case, that specification defines the file extension and required behavior.

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

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.

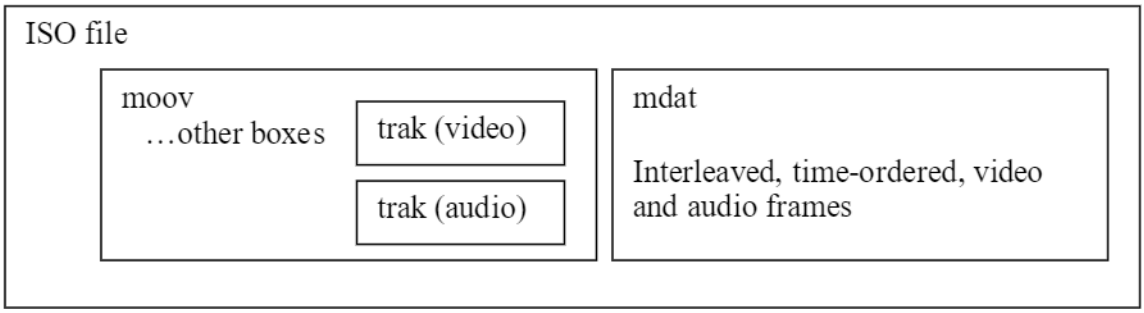

Figure 1 — Simple interchange file

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.

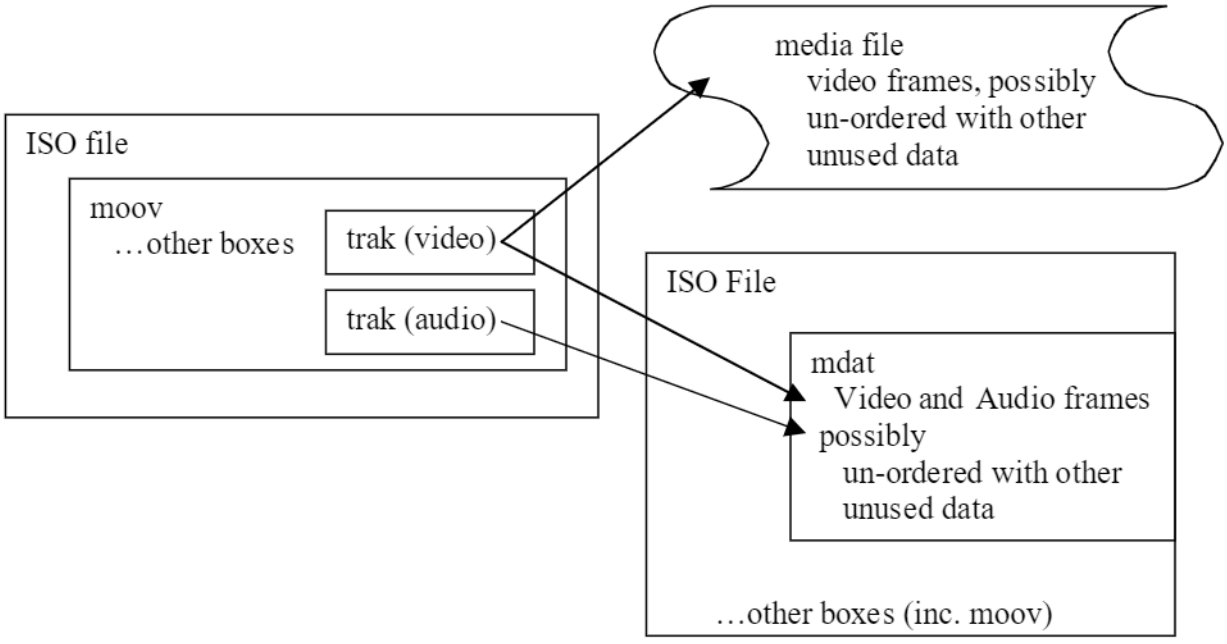

Figure 2 — Content Creation File

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.

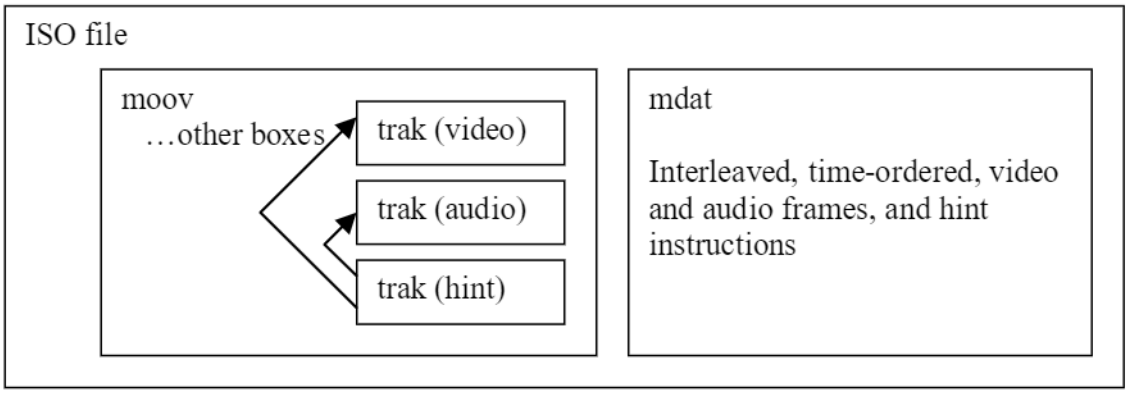

Figure 3 — Hinted Presentation for Streaming

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 favoring 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 behavior (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.

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 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 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 8.33).

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							8.43	*progressive download information*
moov						*	8.1	*container for all the metadata*
	mvhd					*	8.3	*movie header, overall declarations*
	trak					*	8.4	*container for an individual track or stream*
		tkhd				*	8.5	*track header, overall information about the track*
		tref					8.6	*track reference container*
		edts					8.25	*edit list container*
			elst				8.26	*an edit list*
		mdia				*	8.7	*container for the media information in a track*
			mdhd			*	8.8	*media header, overall information about the media*
			hdlr			*	8.9	*handler, declares the media (handler) type*
			minf			*	8.10	*media information container*
				vmhd			8.11.2	*video media header, overall information (video track only)*
				smhd			8.11.3	*sound media header, overall information (sound track only)*
				hmhd			8.11.4	*hint media header, overall information (hint track only)*
				nmhd			8.11.5	*Null media header, overall information (some tracks only)*
				dinf		*	8.12	*data information box, container*
					dref	*	8.13	*data reference box, declares source(s) of media data in track*
				stbl		*	8.14	*sample table box, container for the time/space map*
					stsd	*	8.16	*sample descriptions (codec types, initialization etc.)*
					stts	*	8.15.2	*(decoding) time-to-sample*
					ctts		8.15.3	*(composition) time to sample*
					stsc	*	8.18	*sample-to-chunk, partial data-offset information*
					stsz		8.17.2	*sample sizes (framing)*
					stz2		8.17.3	*compact sample sizes (framing)*
					stco	*	8.19	*chunk offset, partial data-offset information*
					co64		8.19	*64-bit chunk offset*
					stss		8.20	*sync sample table (random access points)*
					stsh		8.21	*shadow sync sample table*
					padb		8.23	*sample padding bits*
					stdp		8.22	*sample degradation priority*
					sdtp		8.40.2	*independent and disposable samples*
					sbgp		8.40.3.2	*sample-to-group*
					sgpd		8.40.3.3	*sample group description*
					subs		8.42	*sub-sample information*
	mvex						8.29	*movie extends box*
		mehd					8.30	*movie extends header box*
		trex				*	8.31	*track extends defaults*
	ipmc						8.45.4	*IPMP Control Box*
moof							8.32	*movie fragment*
	mfhd					*	8.33	*movie fragment header*
	traf						8.34	*track fragment*
		tfhd				*	8.35	*track fragment header*
		trun					8.36	*track fragment run*
		sdtp					8.40.2	*independent and disposable samples*
		sbgp					8.40.3.2	*sample-to-group*
		subs					8.42	*sub-sample information*
mfra							8.37	*movie fragment random access*
	tfra						8.38	*track fragment random access*
	mfro					*	8.39	*movie fragment random access offset*
mdat							8.2	*media data container*
free							8.24	*free space*
skip							8.24	*free space*
	udta						8.27	*user-data*

Table 1 (*continued*)

		cprt					8.28	*copyright etc.*
meta							8.44.1	*metadata*
	hdlr					*	8.9	*handler, declares the metadata (handler) type*
	dinf						8.12	*data information box, container*
		dref					8.13	*data reference box, declares source(s) of metadata items*
	ipmc						8.45.4	*IPMP Control Box*
	iloc						8.44.3	*item location*
	ipro						8.44.5	*item protection*
		sinf					8.45.1	*protection scheme information box*
			frma				8.45.2	*original format box*
			imif				8.45.3	*IPMP Information box*
			schm				8.45.5	*scheme type box*
			schi				8.45.6	*scheme information box*
	iinf						8.44.6	*item information*
	xml						8.44.2	*XML container*
	bxml						8.44.2	*binary XML container*
	pitm						8.44.4	*primary item reference*

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 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 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).

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 server hint tracks; this common format is protocol independent, but contains the declarations of which protocol(s) are described in the server track(s);
- There is a specific design of the server 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 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 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.

The hint tracks are built and flagged so that when the presentation is played back locally (not streamed), they may be ignored.

7.3 Hint Track Format

Hint tracks are used to describe to a server how to serve the elementary stream data in the file over streaming protocols. Each protocol has its own hint track format. The format of the hints is described by the sample description for the hint track. 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 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.

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.
- They are usually marked as disabled for local playback, with their track header 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.

8 Box Definitions

8.1 Movie Box

8.1.1 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.

8.1.2 Syntax

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

8.2 Media Data Box

8.2.1 Definition

Box Type: **'mdat'**
Container: **File**
Mandatory: **No**
Quantity: **Any number**

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 8.14, and the item location box, subclause 8.44.3).

In large presentations, it may be desirable to have more data in this box than a 32-bit size would permit. In this case, the large variant of the size field, above in subclause 6.2, is used.

There may be any number of these boxes in the file (including zero, if all the media data is in other files). The metadata refers to media data by its absolute offset within the file (see subclause 8.19, 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.

8.2.2 Syntax

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

8.2.3 Semantics

`data` is the contained media data

8.3 Movie Header Box

8.3.1 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.

8.3.2 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;
}
```

8.3.3 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.
`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 or larger than all 1s (32-bit maxint), and a new media track is to be added, then a search must be made in the file for a unused track identifier.

8.4 Track Box

8.4.1 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.

8.4.2 Syntax

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

8.5 Track Header Box

8.5.1 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. Hint tracks should have the track header flags set to 0, so that they are ignored for local playback and preview.

8.5.2 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;
}
```

8.5.3 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 (32-bit maxint).
`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.

8.6 Track Reference Box

8.6.1 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.

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.

8.6.2 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[];
}
```

8.6.3 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:

- 'hint' the referenced track(s) contain the original media for this hint track
- 'cdsc' this track describes the referenced track.

8.7 Media Box

8.7.1 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.

8.7.2 Syntax

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

8.8 Media Header Box

8.8.1 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.

8.8.2 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;
}
```

8.8.3 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).
`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.

8.9 Handler Reference Box

8.9.1 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.

8.9.2 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;
}
```

8.9.3 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

handler_type when present in a meta box, contains an appropriate value to indicate the format of the meta box contents

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).

8.10 Media Information Box

8.10.1 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.

8.10.2 Syntax

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

8.11 Media Information Header Boxes

8.11.1 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.

8.11.2 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.

8.11.2.1 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};
}
```

8.11.2.2 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

8.11.3 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.

8.11.3.1 Syntax

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

8.11.3.2 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 center (the normal value); full left is -1.0 and full right is 1.0.

8.11.4 Hint Media Header Box

The hint media header contains general information, independent of the protocol, for hint tracks.

8.11.4.1 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;
}
```

8.11.4.2 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

8.11.5 Null Media Header Box

Streams other than visual and audio may use a null Media Header Box, as defined here.

8.11.5.1 Syntax

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

8.11.5.2 Semantics

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

8.12 Data Information Box

8.12.1 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.

8.12.2 Syntax

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

8.13 Data Reference Box

8.13.1 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.

8.13.2 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;
   }
}
```

8.13.3 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.

8.14 Sample Table Box

8.14.1 Definition

Box Type: **'stbl'**
Container: **Media Information Box ('minf')**
Mandatory: **Yes**
Quantity: **Exactly one**

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

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

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

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

8.14.2 Syntax

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

8.15 Time to Sample Boxes

8.15.1 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

8.15.2 Decoding Time to Sample Box

8.15.2.1 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).

8.15.2.2 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

8.15.2.3 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.

8.15.3 Composition Time to Sample Box

8.15.3.1 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. Since decoding time must be less than the composition time, 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.

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.

8.15.3.2 Syntax

```
aligned(8) class CompositionOffsetBox
   extends FullBox('ctts', 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_offset;
   }
}
```

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

8.15.3.3 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 a non-negative integer that gives the offset between CT and DT, such that CT(n) = DT(n) + CTTS(n).

8.16 Sample Description Box

8.16.1 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. 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.

In audio tracks, the sampling rate of the audio should be used as the timescale of the media, and also documented in the samplerate field here.

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.

8.16.2 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 [];
}

   // Visual Sequences

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;
}


   // 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 = {timescale 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;
      }

   }

}
```

8.16.3 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 either 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

8.17 Sample Size Boxes

8.17.1 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.

8.17.2 Sample Size Box

8.17.2.1 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;
      }
   }
}
```

8.17.2.2 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.

8.17.3 Compact Sample Size Box

8.17.3.1 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;
   }
}
```

8.17.3.2 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.

8.18 Sample To Chunk Box

8.18.1 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.

8.18.2 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;
   }
}
```

8.18.3 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

8.19 Chunk Offset Box

8.19.1 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.

8.19.2 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;
   }
}
```

8.19.3 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.

8.20 Sync Sample Box

8.20.1 Definition

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

This box provides a compact marking of the random access points 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 random access point.

8.20.2 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;
   }
 }
```

8.20.3 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 random access points within the stream and the following table is empty.
`sample_number` gives the numbers of the samples that are random access points in the stream.

8.21 Shadow Sync Sample Box

8.21.1 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 random access 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.

8.21.2 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;
   }
 }
```

8.21.3 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.

8.22 Degradation Priority Box

8.22.1 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.

8.22.2 Syntax

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

8.22.3 Semantics

`version` - is an integer that specifies the version of this box.
`priority` - is integer specifying the degradation priority for each sample.

8.23 Padding Bits Box

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.

8.23.1 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;
   }
 }
```

8.23.2 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

8.24 Free Space Box

8.24.1 Definition

Box Types: 'free', 'skip'
Container: File or other box
Mandatory: No
Quantity: Any number

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).

8.24.2 Syntax

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

8.24.3 Semantics

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

8.25 Edit Box

8.25.1 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.

8.25.2 Syntax

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

8.26 Edit List Box

8.26.1 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 that 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

8.26.2 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;
   }
}
```

8.26.3 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.

8.27 User Data Box

8.27.1 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.

8.27.2 Syntax

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

8.28 Copyright Box

8.28.1 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.

8.28.2 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;
}
```

8.28.3 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.

8.29 Movie Extends Box

8.29.1 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.

8.29.2 Syntax

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

8.30 Movie Extends Header Box

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.

8.30.1 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;
   }
}
```

8.30.2 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.

8.31 Track Extends Box

8.31.1 Definition

Box Type: **'trex'**
Container: **Movie Extends Box ('mvex')**
Mandatory: **Yes**
Quantity: **Exactly one per 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(6)   reserved=0;
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_difference_sample;
      // i.e. when 1 signals a non-key or non-sync sample
unsigned int(16)  sample_degradation_priority;
```

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

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

8.31.2 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
}
```

8.31.3 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.

8.32 Movie Fragment Box

8.32.1 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.

8.32.2 Syntax

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

8.33 Movie Fragment Header Box

8.33.1 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.

8.33.2 Syntax

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

8.33.3 Semantics

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

8.34 Track Fragment Box

8.34.1 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.

8.34.2 Syntax

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

8.35 Track Fragment Header Box

8.35.1 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.

8.35.2 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
}
```

8.35.3 Semantics

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

8.36 Track Fragment Run Box

8.36.1 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).

8.36.2 Syntax

```
aligned(8) class TrackRunBox
         extends FullBox('trun', 0, 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
      unsigned int(32)  sample_composition_time_offset;
   }[ sample_count ]
}
```

8.36.3 Semantics

`sample_count` the number of samples being added in this fragment; 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 set for the first sample only of this run.

8.37 Movie Fragment Random Access Box

8.37.1 Definition

Box Type: **'mfra'**
Container: **File**
Mandatory: **No**
Quantity: **Exactly one**

The Movie Fragment Random Access Box ('mfra') provides a table which may assist readers in finding random access points 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 random access points 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 random access points, incorrect.

8.37.2 Syntax

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

8.38 Track Fragment Random Access Box

8.38.1 Definition

Box Type: 'tfra'
Container: Movie Fragment Random Access Box ('mfra')
Mandatory: No
Quantity: One or more

Each entry contains the location and the presentation time of the random accessible sample. It indicates that the sample in the entry can be random accessed. Note that not every random accessible 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.

8.38.2 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;
   }
}
```

8.38.3 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 random access point and no table entry follows.
`time` is 32 or 64 bits integer that indicates the presentation time of the random access 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 random accessible sample. The number ranges from 1 (the first traf is numbered 1) in each 'moof'.
`trun_number` indicates the 'trun' number that contains the random accessible sample. The number ranges from 1 in each 'traf'.
`sample_number` indicates the sample number that contains the random accessible sample. The number ranges from 1 in each 'trun'.

8.39 Movie Fragment Random Access Offset Box

8.39.1 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.

8.39.2 Syntax

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

8.39.3 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..

8.40 AVC Extensions

8.40.1 Introduction

This section documents technical additions, that were originally designed for AVC support, but which are more generally applicable.

8.40.2 Independent and Disposable Samples Box

8.40.2.1 Definition

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

This optional table answers three questions about sample dependency:

1) does this sample depend on others (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 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 point or random access recovery point, and roll-forward from the sync point 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.

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').

A sample dependency Box may also occur in the track fragment Box.

8.40.2.2 Syntax

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

8.40.2.3 Semantics

`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 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

8.40.3 Sample Groups

8.40.3.1 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.

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.

8.40.3.2 SampleToGroup Box

8.40.3.2.1 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.

8.40.3.2.2 Syntax

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

8.40.3.2.3 Semantics

version is an integer that specifies the version of this box.
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 shall exist for a track.
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 behavior 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.

8.40.3.3 Sample Group Description Box

8.40.3.3.1 Definition

Box Types: 'sgpd'
Container: Sample Table Box ('stbl')
Mandatory: No
Quantity: Zero or more, with one for each SampleToGroup 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*: the base classes for sample group description entries are not boxes and therefore no size is signaled. 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.

8.40.3.3.2 Syntax

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


// Visual Sequence
abstract class VisualSampleGroupEntry (type) extends SampleGroupDescriptionEntry
(type)
{
}


// Audio Sequences
abstract class AudioSampleGroupEntry (type) extends SampleGroupDescriptionEntry
(type)
{
}
```

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

8.40.3.3.3 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.

8.40.3.4 Representation of group structures in Movie Fragments

Support for new SampleGroup 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 8.40.3.2.

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.

8.40.4 Random Access Recovery Points

8.40.4.1 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.

8.40.4.2 Syntax

```
class VisualRollRecoveryEntry() extends VisualSampleGroupEntry ('roll')
{
   signed int(16) roll_distance;
}


class AudioRollRecoveryEntry() extends AudioSampleGroupEntry ('roll')
{
   signed int(16) roll_distance;
}
```

8.40.4.3 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.

8.41 Sample Scale Box

8.41.1 Definition

Box Type: 'stsl'
Container: Sample Entry
Mandatory: No
Quantity: zero or one

This box may be present in any visual sample entry. This box indicates the scaling method that is applied when the width and height of the visual material (as declared by the width and height values in any visual sample entry) do not match the track width and height values (as indicated in the track header box). Implementation of this box is optional; if this box is present and can be interpreted by the decoder, all samples shall be displayed according to the scaling behavior that is specified in this box. Otherwise, all samples are scaled to the size that is indicated by the width and height field in the Track Header Box.

If the size of the image is bigger than the size of the presentation region and 'hidden' scaling is applied in the Sample Scale Box, it is not possible to display the whole image. In such a case, it is useful to provide the information to determine the region that is to be displayed. The center values would then indicate the center of the region of high priority in each visual sample. The decoder can display the region of high priority according to these values. The center values imply a consistent crop for all the images in a sequence. The offset values are positive when the desired visual center is below or to the right of the image center, and negative for offsets above or to the left.

The semantics of the values for scale_method are as specified for the 'fit' attribute of regions in SMIL 1.0.

8.41.2 Syntax

```
aligned(8) class SampleScaleBox extends FullBox('stsl', version = 0, 0)  {
   bit(7) reserved = 0;
   bit(1) constraint_flag;
   unsigned int(8)  scale_method;
   int(16) display_center_x;
   int(16) display_center_y;

}
```

8.41.3 Semantics

`constraint_flag`: if this flag is set, all samples described by this sample entry shall be scaled according to the method specified by the field 'scale_method'. Otherwise, it is recommended that all the samples be scaled according to the method specified by the field 'scale_method', but can be displayed in an implementation dependent way, which may include not scaling the image (i.e. neither to the width and height specified in the track header box, nor by the method indicated here)

`scale_method` is an 8-bit unsigned integer that defines the scaling mode to be used. Of the 256 possible values the values 0 through 127 are reserved for use by ISO and values 128 through 255 are user-defined and are not specified in this International Standard; they may be used as determined by the application. Of the reserved values the following modes are currently defined:

- 1 scaling is done by 'fill' mode.
- 2 scaling is done by 'hidden' mode.
- 3 scaling is done by 'meet' mode.
- 4 scaling is done by 'slice' mode in the x-coordinate.
- 5 scaling is done by 'slice' mode in the y-coordinate.

`display_center_x` is an horizontal offset in pixels of the center of the region that should be displayed by priority relative to the center of the image. Default value is zero. Positive values indicate a display center to the right of the image center.

`display_center_y` is an vertical offset in pixels of the center of the region that should be displayed by priority relative to the center of the image. Default value is zero. Positive values indicate a display center below the image center.

8.42 Sub-Sample Information Box

8.42.1 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.

8.42.2 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;
         }
      }
   }
}
```

8.42.3 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.

8.43 Progressive Download Information Box

8.43.1 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.

8.43.2 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;
   }
}
```

8.43.3 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.

8.44 Metadata Support

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

8.44.1 The Meta box

8.44.1.1 Definition

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

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.

If an ItemProtectionBox 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.

8.44.1.2 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
   Box   other_boxes[];                        // optional
}
```

8.44.1.3 Semantics

The structure or format of the metadata is declared by the handler.

8.44.2 XML Boxes

8.44.2.1 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 BinaryXML 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.

8.44.2.2 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
}
```

8.44.3 The Item Location Box

8.44.3.1 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 values, specifying the size in bytes of the offset field, length field, and base_offset field, respectively. These values must be from the set {0, 4, 8}.

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 file (offset 0) is implied.
- If the length is not specified, or specified as zero, then the entire file length 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; base_offset adds to those offsets.

8.44.3.2 Syntax

```
aligned(8) class ItemLocationBox extends FullBox('iloc', version = 0, 0) {
   unsigned int(4)   offset_size;
   unsigned int(4)   length_size;
   unsigned int(4)   base_offset_size;
   unsigned int(4)   reserved;
   unsigned int(16)  item_count;
   for (i=0; i<item_count; i++) {
      unsigned int(16)  item_ID;
      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++) {
         unsigned int(offset_size*8) extent_offset;
         unsigned int(length_size*8) extent_length;
      }
   }
}
```

8.44.3.3 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.
`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).
`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_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.

8.44.4 Primary Item Box

8.44.4.1 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.

8.44.4.2 Syntax

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

8.44.4.3 Semantics

`item_ID` is the identifier of the primary item

8.44.5 Item Protection Box

8.44.5.1 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.

8.44.5.2 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;
   }
}
```

8.44.6 Item Information Box

8.44.6.1 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.

8.44.6.2 Syntax

```
aligned(8) class ItemInfoEntry
      extends FullBox('infe', version = 0, 0) {
   unsigned int(16)     item_ID;
   unsigned int(16)     item_protection_index
   string         item_name;
   string         content_type;
   string         content_encoding; //optional
}

aligned(8) class ItemInfoBox
      extends FullBox('iinf', version = 0, 0) {
   unsigned int(16) entry_count;
   ItemInfoEntry[ entry_count ]     item_infos;
}
```

8.44.6.3 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;
`content_type` is the MIME type for the item.
`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.
`entry_count` provides a count of the number of entries in the following array.

8.44.7 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.

8.44.8 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.

8.44.8.1 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.

8.44.8.2 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.

8.44.8.3 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.

8.45 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 signaling 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 that 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.

8.45.1 Protection Scheme Information Box

8.45.1.1 Definition

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

The ProtectionSchemeInfoBox 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 ProtectionScheme Info 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 signaling 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) Standalone IPMP: an IPMPInfoBox, when IPMP descriptors outside MPEG-4 systems are used;

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

8.45.1.2 Syntax

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

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

8.45.2 Original Format Box

8.45.2.1 Definition

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

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

8.45.2.2 Syntax

```
aligned(8) class OriginalFormatBox(codingname) extends Box ('frma') {
   unsigned int(32)  data_format = codingname;
                     // format of decrypted, encoded data
}
```

8.45.2.3 Semantics

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

8.45.3 IPMPInfoBox

8.45.3.1 Definition

Box Type: 'imif'
Container: Protection Scheme Information Box ('sinf')
Mandatory: No
Quantity: Exactly One

The IPMPInfoBox contains IPMP Descriptors which document the protection applied to the stream.

IPMP_Descriptor is defined in 14496-1. This is a part of the MPEG-4 object descriptors (OD) that describe how an object can be accessed and decoded. Here, in the ISO Base Media File Format, IPMP Descriptor can be carried directly in IPMPInfoBox without the need for OD stream.

The presence of IPMP Descriptor in this IPMPInfoBox indicates the associated media stream is protected by the IPMP Tool described in the IPMP Descriptor.

Each IPMP_Descriptor has an IPMP_ToolID, which identifies the required IPMP tool for protection. An independent registration authority (RA) is used so any party can register its own IPMP Tool and identify this without collisions.

The IPMP_Descriptor carries IPMP information for one or more IPMP Tool instances, it includes but not limited to IPMP Rights Data, IPMP Key Data, Tool Configuration Data, etc.

More than one IPMP Descriptors can be carried in this IPMPInfoBox if this media stream is protected by more than one IPMP Tools.

8.45.3.2 Syntax

```
aligned (8) class IPMPInfoBox extends FullBox('imif', 0, 0){
   IPMP_Descriptor   ipmp_desc[];
}
```

8.45.3.3 Semantics

`IPMP_descr` is an array of IPMP descriptors

8.45.4 IPMP Control Box

8.45.4.1 Definition

Box Types: 'ipmc'
Container: Movie Box ('moov') or Meta Box ('meta')
Mandatory: No
Quantity: Zero or One

The IPMP Control Box may contain IPMP descriptors which may be referenced by any stream in the file.

The IPMP_ToolListDescriptor is defined in 14496-1, which conveys the list of IPMP tools required to access the media streams in an ISO Base Media File or meta-box, and may include a list of alternate IPMP tools or parametric descriptions of tools required to access the content.

The presence of IPMP Descriptor in this IPMPControlBox indicates that media streams within the file or meta-box are protected by the IPMP Tool described in the IPMP Descriptor. More than one IPMP Descriptors can be carried here, if there are more than one IPMP Tools providing the global governance.

8.45.4.2 Syntax

```
aligned(8) class IPMPControlBox extends FullBox('ipmc', 0, flags) {
   IPMP_ToolListDescriptor toollist;
   int(8)  no_of_IPMPDescriptors;
   IPMP_Descriptor ipmp_desc[no_of_IPMPDescriptors];
   }
```

8.45.4.3 Semantics

`toollist` is an . IPMP_ToolListDescriptor as defined in 14496-1
`no_of_IPMPDEscriptors` is a count of the size of the following array
`ipmp_desc[]` is an array of IPMP descriptors

8.45.5 Scheme Type Box

8.45.5.1 Definition

Box Types: 'schm'
Container: Protection Scheme Information Box ('sinf'), or SRTP Process box ('srpp')
Mandatory: No
Quantity: Exactly one

The SchemeTypeBox ('schm') identifies the protection scheme.

8.45.5.2 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
   }
}
```

8.45.5.3 Semantics

`scheme_type` is the code defining the protection 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.

8.45.6 Scheme Information Box

8.45.6.1 Definition

Box Types: 'schi'
Container: Protection Scheme Information Box ('sinf'), 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 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 SchemeTypeBox.

8.45.6.2 Syntax

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

9 Extensibility

9.1 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.

9.2 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.

9.3 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) VisualSampleEntry, and containing new boxes.

10 RTP and SRTP Hint Track Format

10.1 Introduction

RTP is the real-time streaming protocol defined by the IETF (RFC 1889 and 1890) 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 (e.g. 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 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 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). Hints are not 'composed'. Instead, the hinting process 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.

10.2 Sample Description Format

RTP 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 behavior, 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.

10.2.1 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 SchemeTypeBox and SchemeInformationBox have the syntax defined above for protected media tracks. They serve to provide the parameters required for applying SRTP. The SchemeTypeBox is used to indicate the necessary keymanagement and security policy for the stream in extension to the defined algorithmic pointers provided by the SRTPProcessBox. The keymanagement 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

10.3 Sample Format

Each sample in the 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[];
}
```

10.3.1 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)   reserved;
   bit(1)   P_bit;
   bit(1)   X_bit;
   bit(4)   reserved;
   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];
}

class rtpoffsetTLV() extends Box('rtpo') {
   int(32)  offset;
}
```

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).

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').

The extra-flag indicates that there is extra information before the constructors, in the form of type-length-value sets. Only one such set is currently defined; '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. Note that the extra-information-length is the length in bytes of this field and all the TLV entries. Note also that the TLV boxes are aligned on 32-bit boundaries; the box size indicates the actual bytes used, not the padded length. The extra-information-length will be correct.

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.

10.3.2 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).

10.4 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).

10.4.1 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.

10.4.2 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.

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

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 90° 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 random access point. Locating the nearest random access point requires consulting two boxes. 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. The shadow sync box gives the opportunity for a content author to provide samples that are not delivered in the normal course of delivery, but which can be inserted to provide additional random access points. This improves random access without impacting bitrate during normal delivery. This box maps samples that are not random access points to alternate samples that are. You should also consult this table if present to find the first shadow sync sample prior to the sample in question. Having consulted the sync sample table and the shadow sync 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 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 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.

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
2.	Matsushita Electric Industrial Co., Ltd
3.	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 | 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. So there will be a number of specifications depending on the codecs, combination of codecs, and so on.

C.2 General Principles

A number of existing file formats use the ISO Base Media File Format, not least the MPEG-4 MP4 File Format, and the Motion JPEG 2000 MJ2 File Format. 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. 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.

C.3 Brand Identifiers

C.3.1 Introduction

This section covers the use of brand identifiers in the file-type box, including:

- Introduction of a new brand.
- Player's behavior depending on the brand.
- Setting of the brand on the creation of the ISO Base Media file.

C.3.2 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 in the File Type Box. In the File Type Box, two kinds of brands can be indicated. One is the major_brand that identifies the specification of the best use for the file. Second is the compatible_brands, which can identify multiple specifications to which the file complies.

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.

C.3.3 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.

If a new brand is defined, it should not be required that it be a major-brand. However, brands that are only permitted as compatible brands may be defined.

C.3.4 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.3.5 Authoring Guideline

If the author wants to create a file that complies to 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.3.6 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 to '3gp5' and does not support MP3, recommended behavior of the player is to play only H.263. If the content's author does not expect such behavior, 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 behavior and the file is played only when the player supports both H.263 and MP3.

C.4 Box layout and order

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.

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.

C.6 Use of Template fields

Template fields are defined in the file format. If any are used in a derived specification, the use must be compatible with the base definition, and that use explicitly documented.

C.7 Construction of fragmented movies

When constructing a fragmented file for playback, there are some recommendations for structuring the content which would optimize playback and random access. The recommendations are as follows:

- The file should consist of boxes in the following order:
  - 'ftyp'

- 'moov'
- pair of 'moof' and 'mdat' (arbitrary number)
- 'mfra'

- A 'moof' box consists of at most one 'traf' for each media. When the file contains a single video track and a single audio track, the 'moof' will contain two 'traf', one for the video and one for the audio.
- For video, random accessible samples are stored as the first sample of each 'traf'. In the case of gradual decoder refresh, a random accessible sample and the corresponding recovery point are stored in the same movie fragment. For audio, samples having the closest presentation time for every video random accessible sample are stored as the first sample of each 'traf'.Hence, the first samples of each media in the 'moof' have the approximately equal presentation times.
- First (random accessible) samples are recorded in the 'mfra' for both video and audio.
- All samples in 'mdat' are interleaved with an appropriate interleave depth.

The offset and the initial presentation time of every 'moof' are given in the 'mfra' for both audio and video.

The player will load the 'moov' and 'mfra' initially, and hold them in memory during playback. When random access is needed, the player will search 'mfra' in order to find the random access point having the closest presentation time for the indicated time.

Since the first sample in the 'moof' is random accessible, the player can directory jump in on the random access point. The player can read the 'moof' of the random access point from the beginning. The subsequent 'mdat' starts from the random accessible sample. As such, a two-step seeking would not be necessary for random access.

Note that an 'mfra' box is optional and may never occur.

Annex D
(informative)

Registration Authority

D.1 Code points to be registered

The code-points within the file format are all 32-bit fields, normally four printable characters (commonly known as four-character-codes or 4CCs). An objecttype identifier is an 8-bit integer.

The code-points that may be registered are:

1) File format box identifiers. Note that in some specifications boxes were known as atoms. Note that the introduction of new atom types is discouraged; in general other extensibility features of the file format should be used if possible.

2) File format track type identifiers. A pair of identifiers is usually used here, to identify the track type (audio, video, etc.) and, if required, a media-specific header atom (video media header, etc.). It is expected that the need for new track types is rare, however; most media should fall into existing types (e.g. video codecs should use video tracks, hint protocols use hint tracks, and so on).

3) File format sample description and sample format identifiers (also known as codec names). This includes audio and video codecs, and also protocol identifiers for hint tracks. Any registration of a new sample format will automatically be issued an object-type identifier also (see below), thus making the identification of the carriage of this format within the MPEG-4 systems object descriptor framework possible.

4) File format track reference identifiers. Dependencies between tracks are typed in the file format (for example, hint tracks depend on the media tracks they hint, using a track dependency of type 'hint').

5) This specification includes a 'file type' atom which includes a list of 'brands' which identify which specifications the file is conformant to. Bodies defining standards based on the structural definition of this file format would normally use a new brand to identify files conformant to their specification. Any registration of a new brand must specify the precise specification which the brand identifies.

6) Within the MPEG-4 object descriptor framework, the objecttype value is used to identify the format of the streams. An objecttype identifier may be requested independently of the file format identifiers above.

7) Sample groups associate typed information with groups of samples. The grouping type may be registered.

8) Both media and metadata can be protected and the protection scheme used identified with a registered protection scheme type.

These code-points are referred to in the rest of this annex as registered identifiers, abbreviated as RIDs.

D.2 Procedure for the request of an MPEG-4 registered identifier value

Requesters of an MPEG-4 code-points as detailed above value to identify a private data format shall apply to the Registration Authority. Registration forms shall be available from the Registration Authority. The requester shall provide the information specified in D.4. Companies and organizations are eligible to apply.

D.3 Responsibilities of the Registration Authority

The primary responsibilities of the Registration Authority administrating the registration of the private data format identifiers are outlined in this annex; certain other responsibilities may be found in the JTC 1 Directives. The Registration Authority shall:

a) implement a registration procedure for application for a unique RID in accordance with the JTC 1 Directives;

b) receive and process the applications for allocation of an identifier from application providers;

c) ascertain which applications received are in accordance with this registration procedure, and to inform the requester within 30 days of receipt of the application of their assigned RID;

d) inform application providers whose request is denied in writing with 30 days of receipt of the application, and to consider resubmissions of the application in a timely manner;

e) maintain an accurate register of the allocated identifiers. Revisions to format specifications shall be accepted and maintained by the Registration Authority;

f) make the contents of this register available upon request to National Bodies of JTC 1 that are members of ISO or IEC, to liaison organizations of ISO or IEC and to any interested party;

g) maintain a data base of RID request forms, granted and denied. Parties seeking technical information on the format of private data which has a RID shall have access to such information which is part of the data base maintained by the Registration Authority;

h) report its activities annually to JTC 1, the ITTF, and the SC 29 Secretariat, or their respective designees; and

i) accommodate the use of existing RIDs whenever possible.

D.4 Contact information for the Registration Authority

Apple Computer Inc.

One Infinite Loop, M/S 301-4B
Cupertino, California 95014
USA
E-mail: mp4reg@group.apple.com
Web: http://www.mp4ra.org/

D.5 Responsibilities of Parties Requesting a RID

The party requesting a format identifier shall:

a) apply using the Form and procedures supplied by the Registration Authority;

b) include a description of the purpose of the registered identifier, and the required technical details as specified in the application form;

c) provide contact information describing how a complete description can be obtained on a non-discriminatory basis;

d) agree to institute the intended use of the granted RID within a reasonable time frame; and

e) to maintain a permanent record of the application form and the notification received from the Registration Authority of a granted RID.

D.6 Appeal Procedure for Denied Applications

The Registration Management Group is formed to have jurisdiction over appeals to denied request for a RID. The RMG shall have a membership who is nominated by P- and L-members of the ISO technical committee responsible for ISO/IEC 14496. It shall have a convenor and secretariat nominated from its members. The Registration Authority is entitled to nominate one non-voting observing member.

The responsibilities of the RMG shall be:

a) to review and act on all appeals within a reasonable time frame;

b) to inform, in writing, organizations which make an appeal for reconsideration of its petition of the RMGs disposition of the matter;

c) to review the annual report of the Registration Authorities summary of activities; and

d) to supply Member Bodies of ISO and National Committees of IEC with information concerning the scope of operation of the Registration Authority.

D.7 Registration Application Form

D.7.1 Contact Information of organization requesting a RID

Organization Name:

Address:

Telephone:

Fax:

E-mail:

Telex:

D.7.2 Request for a specific RID

NOTE — If the system has already been implemented and is in use, fill in this item and item D.7.3D.7.3 and skip to D.7.5, otherwise leave this space blank and skip to D.7.3)

D.7.3 Short description of RID that is in use and date system was implemented

D.7.4 Statement of an intention to apply the assigned RID

D.7.5 Date of intended implementation of the RID

D.7.6 Authorized representative

Name:

Title:

Address:

Email:

Signature _____

D.7.7 For official use of the Registration Authority

Registration Rejected _____

Reason for rejection of the application:

Registration Granted _____ Registration Value _____

Attachment 1 — Attachment of technical details of the registered data format.

Attachment 2 — Attachment of notification of appeal procedure for rejected applications.

Bibliography

The QuickTime file format specification, in PDF:

<http://www.apple.com/quicktime/resources/qtfileformat.pdf>

ICS 35.040

Price based on 84 pages