ISO/IEC JTC 1/SC 29/WG1 N2117

Date: 27-March-2001

ISO/IEC JTC 1/SC 29/WG 1
(ITU-T SG8)

Coding of Still Pictures

JBIG	JPEG
Joint Bi-level Image Experts Group	Joint Photographic Experts Group

TITLE:	Motion JPEG2000 Final Committee Draft 1.0
SOURCE:	Takahiro Fukuhara, Sony Corporation David Singer, Apple Computer
PROJECT:	ISO/IEC 15444-3 (JPEG2000, Part 3)
STATUS:	
REQUESTED ACTION:	Request for National Body comments.
DISTRIBUTION:	WG1 website and distribute to WG1

Contact:
ISO/IEC JTC 1/SC 29/WG 1 Convener - Dr. Daniel T. Lee
Yahoo! 3420 Central Expressway, Santa Clara, California 95051, USA
Tel: +1 408 992 7051, Fax: +1 253 830 0372, E-mail: dlee@yahoo-inc.com

Table of Contents

1 Introduction

This document specifies the use of the wavelet-based JPEG2000 codec for the coding and display of timed sequences of images. It has been defined by ISO/IEC JTC1 SC29/WG1 as part three of the new JPEG2000 International Standard. In this specification, a file format is defined, and guidelines for the use of the JPEG2000 codec for motion sequences are supplied. The Motion JPEG2000 file format MJ2 is designed to contain one or more motion sequences of JPEG2000 images, with their timing, and also optional audio annotations, all composed into an overall presentation.

Motion JPEG2000 is expected to be used in a variety of applications, particularly where the codec is already available for other reasons, or where the high-quality frame-based approach, with no inter-frame coding, is appropriate. These application areas include:

- digital still cameras,
- error-prone environments such as wireless and the internet,
- PC-based video capturing,
- high quality digital video recording for professional broadcasting and motion picture production from film-based to digital systems,
- remote surveillance,
- and high-resolution medical and satellite imaging.

Motion JPEG2000 is a flexible format, permitting a wide variety of usages, such as editing, display, interchange, and streaming.

1.1 Compatibility and Technology derivation (Normative)

This is a standalone specification; it defines the file format for MJ2. However, it stands as a member of a family of specifications with common formatting.

The other family members include:

- The JPEG2000 single image format, JP2.
- The MPEG-4 file format, MP4.
- The QuickTime file format, on which MP4 and this specification are based.

These specifications share a common definition for the structure of a file (a sequence of objects, called boxes here and atoms in MP4 and QuickTime), and a common definition of the general structure of an object (the size and type).

Note that all these specifications require that objects that are unrecognized by a particular reader shall be ignored.

This specification takes precedence over those from which it inherits, in any case where there are differences or conflicts; however no such conflicts are known to exist.

1.1.1 JP2 Inheritance and Compatibility

The still image format, JP2, defines a number of boxes. The following boxes from that specification must be present. If the JP2 specification requires a particular position (e.g. first in the file), that positioning must be followed here:

1) The JP2 'family' signature box ‘jP2 ‘.
2) The file type compatibility box ‘ftyp’.

In the file type compatibility box, the brand shall be 'mjp2' for files conforming to this specification, and 'mjp2' must be a member of the compatibility list.

It is permissible under this specification to make a file which adheres to both this specification and the JP2 specification. In that case:

1) The compatibility list shall include all the compatible brands
2) The objects (boxes or atoms) required by that specification shall also be present.
3) The objects (boxes or atoms) optional in that specification may also be present.

A still image reader, reading a file which contains both a presentation (conformant to this specification) and a still image, would 'see' only the still image. Likewise a motion reader would 'see' only the presentation. A more powerful reader may display both, or offer the user a choice.

Note that the JP2 specification includes an optional IPR (Intellectual Property Rights) box which is therefore also optional in this specification. Among other issues this addresses unique identification and protection of content.

1.1.2 Conformance

Implementations of Motion JPEG2000 decoders must support JPEG2000 image sequences, as well as raw and twos-complement audio if audio output is available. They may also support compressed audio, using MP4 formats, or other track types from MPEG-4. The support of such MPEG-4 tracks is not required; however, readers must not fail if they are present. If MPEG-4 composition (BIFS) is used, then the simple composition used in this specification should also be set up in such a way that a reader not implementing BIFS will display a suitable result.

Files conformant with this specification must contain at least one Motion JPEG2000 video track. They may contain more video tracks, uncompressed audio, or compressed MP4 audio.

1.1.3 Profiles and Levels

There are two tools for profiling Motion JPEG2000 files.

The first consists of the optional specification of tools and levels of the JPEG2000 coding system (codestream features). These are indicated in the optional sample description extension JP2 Profile Box (see below 6.2.16).

The second tool allows a file overall to be identified as belonging to a definition which forms a proper subset of the general specification. Such definitions might restrict such features as:

- the use of data references, and multiple files
- the layout order of the boxes, and the data within the boxes (e.g. that data is in time order and interleaved)
- the use of profiles of the JPEG2000 codestream
- the existence of other tracks, and their format (e.g. audio, MPEG-7, etc.).

The conformance to these restricted profiles is indicated in the file type box by the addition of the compatible profiles as brands within the compatibility list. The Annex "File and Codestream profiles" defines the available profiles.

1.2 Design principles (Informative)

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 which give the size, type and position of media data units is 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 meta-data is used to describe the media data by reference, not by inclusion.

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.

1.3 MP4 Inheritance and Compatibility (Informative)

Motion JPEG2000 is represented as a peer coding system to MPEG4 visual, in this specification. Data structures and concepts which are held in common with these other specifications are defined to be compatible with them. Most boxes (atoms in MP4) are defined identically; this includes:

> Movie, Media Data, Track, Track Reference, Media, Media Header, Handler Reference, Media Information, Hint Media Header, Data Information, Data Reference, Sample Table, Time to Sample, Sample Size, Sample to Chunk, Chunk Offset, Free Space, Edit, Edit List, User Data, and Extension (UUID) boxes.

A number of boxes are used in a compatible fashion, but there are a number of fields in MP4 which, in that specification, have required initial values but are ignored on reading, which are used here. This includes:

> Movie Header, Track Header, Video Media Header, Sound Media Header,

The format of the Sample Description Box itself is the same, but a new VideoSampleDescription Box for motion JPEG2000 is introduced within it; and likewise, a new Audio Sample Description format for raw audio is introduced.

2 Scope

This document specifies the use of the wavelet-based JPEG2000 codec for the coding and display of timed sequences of images (motion sequences), possibly combined with audio, and composed into an overall presentation. In this specification, a file format is defined, and guidelines for the use of the JPEG2000 codec for motion sequences are supplied.

3 Normative references

1. **The JPEG2000 specification,** ISO/IEC 15444-1.
2. **The MPEG-4 specification,** ISO/IEC 14496-1 :2001, particularly the MP4 file format: section 13, and the syntax description language (SDL), section 14.

4 Additional references

The QuickTime file format specification, in PDF:
<http://www.apple.com/quicktime/resources/qtfileformat.pdf>

5 Definitions (Informative)

Box	An object-oriented building block defined by a unique type identifier and length
Chunk	A contiguous set of samples for one track.
Container Box	A box whose sole purpose is to contain and group a set of related boxes.
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.
Hinter	A tool that is run on a completed file to add one or more hint tracks to the file and so facilitate streaming.

Movie Box	A container box whose sub-boxes define the meta-data for a presentation. (‘moov’).
Movie Data Box	A container box which can hold the actual media data for a presentation (‘mdat’).
Motion sequence	A timed sequence of JPEG2000 images.
MJ2 File	The name of the file format described in this specification.
Presentation	One or more motion sequences (q.v.), possibly combined with audio.
Sample	In non-hint tracks, a sample is an individual frame of video, or a compressed frame of audio. In hint tracks, a sample defines the formation of one or more streaming packets.
Sample Table	A packed directory for the timing and physical layout of the samples in a track.
Track	A collection of related samples (q.v.) in an MJ2 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.

6 File organization (Normative)

6.1 Presentation structure

6.1.1 File Structure

A presentation may be contained in several files. One file contains the meta-data 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 meta-data defined here.

Note that these other files may be MJ2 files, JPEG2000 image files, MPEG-4 files containing JPEG2000 images, or other formats. Only the JPEG2000 images are stored in these other files; all timing and framing (position and size) information is in the MJ2 file, so the ancillary files are essentially free-format.

If an MJ2 file contains hint tracks, the media tracks which reference the media data from which the hints were built must remain in the file, even if the data within them is not directly referenced by the hint tracks.

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 must contain exactly one presentation meta-data wrapper (the Movie Box). It is usually at, or close to, the beginning or end of the file, to permit its easy location. The other objects found at this level may be free space, or media data boxes.

The fields in the objects are stored in network byte order (big-endian format).

6.1.3 Meta Data and Media Data

The meta-data is contained within the meta-data 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 MJ2 file are unique within that file; no two tracks may use the same identifier.

The next track identifier value in the movie header 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 a free track identifier is needed for all additions.

6.1.5 Visual Composition

Composition of multiple image sequences in a 2D environment may be achieved by using multiple video tracks which overlap in time. Their composition is defined by the following structures:

- The matrix in the track header specifies their positioning and scaling.
- The layer field in the track header specifies the front-to-back ordering of the tracks.
- The graphics mode and opcolor fields in the video media header are used to specify the ways in which each track is composited onto the existing image (this compositing is performed from back to front).

Note that applications requiring more complex compositing may use the BIFS system from MPEG-4, optionally. As noted above, the matrix, graphics mode, and layers should be setup so that a reader not implementing BIFS displays the desired result. Matrix values that occur in the headers specify a transformation of video images for presentation. 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
```

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. For upwards compatibility into the MPEG-4 BIFS (scene composition) system, matrices used here should restrict (u,v,w) to be (0,0,1), for which the hex values are (0,0,0x40000000). This permits the simple composition used here to be mapped into BIFS if a scene later requires full scene management.

Tracks are composed to the presentation surface from back (highest layer number) to front (lowest layer number), against an indeterminate initial color. There are various composition modes available; the backmost (first-rendered) track should normally use 'copy' as the initial image is indeterminate. Subsequent layers can then be composed on top in a variety of ways. The following table details the composition modes available. Note that (currently) only the 'transparent' mode uses the opcolor field.

Mode	Code	Description
Copy	0x0	Copy the source image over the destination
Transparent	0x24	Replace the destination pixel with the source pixel if the source pixel isn't equal to the opcolor. (Also known as 'blue-screen').
Alpha	0x100	Replace the destination pixel with a blend of the source and destination pixels, with the proportion controlled by the alpha channel
Premultiplied white alpha	0x101	Pre-multiplied with white means that the color components of each pixel have already been blended with a white pixel, based on their alpha channel value. Effectively, this means that the image has already been combined with a white background, which must be removed before composition.
Premultiplied black alpha	0x102	Pre-multipled with black is the same as pre-multipled with white, except the background color that the image has been blended with is black instead of white.

Table 1 Graphics Composition Modes

6.2 Meta-data Structure (Objects)

The following represents the subset of the QuickTime file specification that is required to define an MJ2 file. 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) may need the 64-bit size.

Note that 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 2 in section 3). Comments in the code indicate informative material.

NOTE: All indexes start with the value one.

```
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 large size; 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'.

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; track reference types tmcd, chap, sync, scpt, ssrc.

Boxes not explicitly defined in this standard may be ignored.

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

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.

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

Fields shown as “pre-defined” in the box descriptions should be initialized to the given value on box creation, copied un-inspected when boxes are copied, and ignored on reading.

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

The table shows boxes which may occur at the top-level in the left-most column; indentation is used to show possible containment. Thus, for example, a track header (tkhd) is found in a track (trak), which is found in a movie (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.

Note that user data objects may be found 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 should be followed for the order of boxes:

1) The JP2 Signature Box and File Type Box **must** occur first and second in the file (see section 1.1.1).

2) All header boxes **should** 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 **must** be in sequence order (see section 6.2.27).

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) The Track Reference Box and Edit List (if any) **should** precede the Media Box, and the Handler Box **should** precede the Media Information Box, and the Data Information Box **should** precede the Sample Table Box.

6) User Data Boxes **should** be placed last in their container, which is either the Movie Box or Track Box.

jP2						*	1.1.1	*the JP2 family signature*
ftyp						*	1.1.1	*file type and compatibility*
moov						*	6.2.1	*container for all the meta-data*
	mvhd					*	6.2.3	*movie header, overall declarations*
	trak						6.2.4	*container for an individual track or stream*
		tkhd				*	6.2.5	*track header, overall information about the track*
		tref					6.2.6	*track reference container*
		edts					6.2.21	*edit list container*
			elst				6.2.22	*an edit list*
		mdia				*	**Error! Reference source not found.**	*container for the media information in a track*
			mdhd			*	6.2.8	*media header, overall information about the media*
			hdlr			*	6.2.9	*handler, declares the media (handler) type*
			minf			*	6.2.10	*media information container*
				vmhd			6.2.11.1	*video media header, overall information (video track only)*
				smhd			6.2.11.2	*sound media header, overall information (sound track only)*
				hmhd			6.2.11.3	*hint media header, overall information (hint track only)*
				dinf		*	**Error! Reference source not found.**	*data information box, container*
					dref	*	0	*data reference box, declares source(s) of media in track*
				stbl		*	**Error! Reference source not found.**	*sample table box, container for the time/space map*
					stsd	*	**Error! Reference source not found.**	*sample descriptions (codec types, initialization etc.)*
					stts	*	6.2.15.1	*(decoding) time-to-sample*
					stsc	*	6.2.18	*sample-to-chunk, partial data-offset information*
					stsz	*	6.2.17	*sample sizes (framing)*
					stco	*	6.2.19	*chunk offset, partial data-offset information*
	mvex						6.2.24	*movie extends box*
		trex				*	0	*track extends defaults*

moof							**Error! Reference source not found.**	*movie fragment*
	mfhd					*	0	*movie fragment header*
	traf						**Error! Reference source not found.**	*track fragment*
		tfhd				*	0	*track fragment header*
		trun					6.2.30	*track fragment run*
mdat							6.2.2	*Media data container*
free							**Error! Reference source not found.**	*free space*
skip							**Error! Reference source not found.**	*free space*
	udta						6.2.23	*user-data, copyright etc.*

Table 2 Box types, structure, and cross-reference

6.2.1 Movie Box

Box Type:	‘moov’
Container:	File
Mandatory:	Yes
Quantity:	Exactly one

The meta-data for a presentation is stored in the single Movie Box which occurs at the top-level of a file. Normally this box is first or last in the sequence of boxes in a file, though this is not required.

6.2.1.1 Syntax

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

6.2.2 Media Data Box

Box Type:	‘mdat’
Container:	File
Mandatory:	No
Quantity:	Any number

This box contains the media data. In video tracks, this box would contain JPEG2000 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 meta-data (see particularly the sample table, section **Error! Reference source not found.**).

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 section 6.2, is used.

Note that there may be any number of these boxes in the file (including zero, if all the media data is in other files). The meta-data refers to media data by its absolute offset within the file (see 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.

6.2.2.1 Syntax

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

6.2.2.2 Semantics

`data` is the contained media data

6.2.3 Movie Header Box

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.

6.2.3.1 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;
      }
      int(32)     rate;       // typically 1.0
      int(16)     volume;     // typically 0x0100, for full volume
      const bit(16)     reserved = 0;
      const unsigned int(32)[2]     reserved = 0;
      int(32)[9]  matrix;
      // unity is { 0x00010000,0,0,0,0x00010000,0,0,0,0x40000000 };
      bit(32)[6]  pre-defined = 0;
      unsigned int(32)  next-track-ID;
}
```

6.2.3.2 Semantics

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

`creation-time` is an integer which declares the creation time of the presentation (in seconds since midnight, Jan. 1, 1904

`modification-time` is an integer which declares the most recent time the presentation was modified (in seconds since midnight, Jan. 1, 1904

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

`duration` is an integer which declares length of the presentation (in the indicated timescale). Note that 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 which indicates the preferred rate to play the presentation; 1.0 (0x10000) is normal forward playback

`volume` is a fixed point 8.8 number which indicates the preferred playback volume. 1.0 (0x100) is full volume.

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

`next-track-ID` is an integer which indicates a value to use for the track ID of the next track to be added to this presentation. Note that 0 is not a valid track ID value. This must be larger than the largest track-ID in use. If this value is equal to or larger than 65535, and a new media track is to be added, then a search must be made in the file for a free track identifier which will fit into 16 bits. If the value is all 1s (32-bit maxint), then this search is needed for all additions.

6.2.4 Track Box

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 may consist 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 must be at least one media track within a MJ2 file, and all the media tracks which contributed to the hint tracks must 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 must remain.

6.2.4.1 Syntax

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

6.2.5 Track Header Box

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 immediately. An empty edit is used to offset the start time of a track.

6.2.5.1 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;
      int(16) layer;
      int(16) pre-defined = 0;
      int(16)     volume;
      const unsigned int(16)  reserved = 0;
      int(32)[9]  matrix;
      // unity is { 0x00010000,0,0,0,0x00010000,0,0,0,0x40000000 };
      int(32)     width;
      int(32)     height;
}
```

6.2.5.2 Semantics

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

`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 movie. Flag value is 0x0002.

Track in preview Indicates that the track is used in the movie's preview. Flag value is 0x0004.

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

`modification-time` is an integer which declares the most recent time the track was modified (in seconds since midnight, Jan. 1, 1904)
`track-ID` is an integer which 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). Note that this property is derived from the track's edits. 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 movie timescale.
`layer` specifies the front-to-back ordering of video tracks; lower numbers are further forward. 0 is the normal value, and -1 would be in front of track 0, and so on.
`volume` is a fixed 8.8 value specifying the track's relative audio volume. 1.0 (0x100) is the normal value. Its value if irrelevant for a purely visual track.
`matrix` provides a transformation matrix for the video; note that (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. Note that these need not be the same as the pixel dimensions of the images, which is documented in the sample description (and the codestream); scaling may occur. This scaling occurs before any overall transformation of the track represented by the matrix.

6.2.6 Track Reference Box

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. In particular, a 'hint' reference links from the containing hint track to the media data which it hints. 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. Note that the reference array is sized to fill the reference type box.

6.2.6.1 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[];
}
```

6.2.6.2 Semantics

The Track Reference Box contains track reference type boxes.

The `reference-type` must be set to one of the following values:

`'hint'`	the referenced track(s) contain the original media for this hint track

6.2.7 Media Box

Box Type:	'mdia'
Container:	Track Box ('trak')
Mandatory:	Yes
Quantity:	Exactly one

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

6.2.7.1 Syntax

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

6.2.8 Media Header Box

Box Type:	'mdhd'
Container:	Media Box ('mdia')
Mandatory:	Yes
Quantity:	Exactly one

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

6.2.8.1 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;
}
```

6.2.8.2 Semantics

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

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

`modification-time` is an integer which declares the most recent time the presentation was modified (in seconds since midnight, Jan. 1, 1904)

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

`duration` is an integer which declares length 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. The code is confined to being three lower-case letters, so these values are strictly positive.

6.2.9 Handler Reference Box

Box Type:	'hdlr'
Container:	Media Box ('mdia')
Mandatory:	Yes
Quantity:	Exactly one

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

6.2.9.1 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;
}
```

6.2.9.2 Semantics

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

`handler-type` is an integer containing one of the following values:

`'vide'`	Video track
`'soun'`	Audio track
`'hint'`	Hint track

`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).

6.2.10 Media Information Box

Box Type:	'minf'
Container:	Media Box ('mdia')
Mandatory:	Yes
Quantity:	Exactly one

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

6.2.10.1 Syntax

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

6.2.11 Media Information Header Boxes

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

There is a media information header for each track type (corresponding to the media handler type).

6.2.11.1 Video Media Header Box

The video media header contains general presentation information, independent of the coding, for video media.

6.2.11.1.1 Syntax

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

6.2.11.1.2 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:

copy = 0x00	copy over the existing image
transparent = 0x24	'blue-screen' this image using the opcolor
alpha = 0x100	alpha-blend this image
whitealpha = 0x101	alpha-blend this image, which has been blended with white

blackalpha = 0x102 alpha-blend this image, which has been blended with black

`opcolor` is a set of 3 color values (red, green, blue) available for use by graphics modes

6.2.11.2 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.

6.2.11.2.1 Syntax

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

6.2.11.2.2 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.

6.2.11.3 Hint Media Header Box

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

6.2.11.3.1 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)  slidingavgbitrate;
}
```

6.2.11.3.2 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

`slidingavgbitrate` gives the maximum rate in bits/second over any window of one minute (corresponding to the avgBitrate field in an MPEG-4 DecoderConfigDescriptor)

6.2.12 Data Information Box

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

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

6.2.12.1 Syntax

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

6.2.13 Data Reference Box

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) which 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 samples. 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) should be supplied in the entry field.

6.2.13.1 Syntax

```
aligned(8) class DataEntryUrlBox
      extends FullBox(‘url ’, version = 0, 0) {
      string      location;
}

aligned(8) class DataEntryUrnBox
      extends FullBox(‘urn ’, version = 0, 0) {
      string      name;
      string      location;
}

aligned(8) class DataReferenceBox
      extends FullBox(‘dref’, version = 0, 0) {
      unsigned int(32)  entry-count;
      for (i=0; i < entry-count; i++) {
            DataEntryBox(entry-version, entry-flags)  data-entry;
      }
}
```

6.2.13.2 Semantics

`version` is an integer that specifies the version of this box
`entry-count` is an integer which 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 which delivers a file (e.g. URLs of type file, http, ftp etc.), which ideally also permits random access. Relative URLs are permissible and are relative to the file containing the movie box which contains this data reference.

6.2.14 Sample Table Box

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 must 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.

6.2.14.1 Syntax

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

6.2.15 Time to Sample Box

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

This box contains a compact version of a table which allows indexing from 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 Time to Sample Box contains time deltas: DT(n+1) = DT(n) + STTS(n) where STTS(n) is the (uncompressed) table entry for sample n.

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 value if it is non-empty (non-zero).

6.2.15.1 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;
            int(32)     sample-delta;
      }
}
```

6.2.15.2 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 which have the given duration
`sample-delta` is an integer that gives the delta of these samples in the time-scale of the media

6.2.16 Sample Description Box

Box Types:	'stsd', 'mjp2', 'raw ', 'twos'
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 data array is track-type specific, and may 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.

A Motion JPEG2000 visual sample entry shall contain a JP2 Header Box from the JPEG2000 part 1 specification; however, the YCbCr enumerated color space (codepoint 22) from Part 2 may also be used to identify the color space used, in addition to the Part 1 enumerated color spaces (sRGB and grayscale).

The sample format for Motion JPEG2000 data is a set of boxes. Currently this specification permits only JP2 Codestream Boxes ('jp2c') as defined in the JP2 specification. If there is no Field Coding Box present, or the field count is 1, the sample shall contain precisely one codestream box. If the field count is 2 then there shall be two codestream boxes. Other boxes, if present in the sample, shall be ignored. The actual codestreams presented to the decoder are formed by concatenating the contents of the JP2 Prefix Box, if any, in the sample description before each codestream presented in the jp2c box(es) in the samples. If field coding is used, the same prefix is concatenated before both fields. Typically, the prefix will contain a JPEG2000 main header; however, this is not required in the general case, though specific profiles may limit the use of the prefix box.

If the codestreams used in a sequence conform to a specific profile of the JPEG2000 coder, a JP2 Profile Box may be used to indicate such conformance.

The visual sample entry may optionally contain a field-ordering box (see below). Note that if fieldcount is 2, each field will be half the height of the overall image, as declared in the 'height' field of the sample description. To be precise, if the height field contains the value H, then the field with the topmost scanline has ((H+1) div 2) lines, and the other field has (H div 2) lines.

For audio tracks, in the formats defined here (with code points 'raw ' and 'twos'), the data is stored as uncompressed samples. If stereo is stored, the data consists of interleaved left/right samples. The raw format uses offset-binary; for 8-bit samples, values range from 0 to 255 with 128 indicating silence. For ‘twos’, values range from -127 to 128, with 0 being silence.

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.

Note that multiple descriptions may be used within a track.

6.2.16.1 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() extends SampleEntry ('mjp2'){
      unsigned int(16) pre-defined = 0;
      const unsigned int(16) reserved = 0;
      unsigned int(32)[3]     pre-defined = 0;
      int(16)     width;
      int(16)     height;
      int(32)     horizresolution;  // typically 72 (0x00480000)
      int(32)     vertresolution;   // typically 72 (0x00480000)
      const unsigned int(32)  reserved = 0;
      unsigned int(16)  pre-defined = 1;
      string[32]  compressorname;
      int(16)     depth;
      int(16)     pre-defined = -1;

      JP2HeaderBox();
      FieldCodingBox(); // optional
      JP2ProfileBox();  // optional
      JP2PrefixBox();   // optional
      JP2SubSamplingBox();    // optional
}
      // Field-Based Coding
class FieldCodingBox() extends Box('fiel'){
      int(8)      fieldcount; // must be 1 or 2
      int(8)      fieldorder; // both storage and temporal order
}

class JP2ProfileBox() extends FullBox('jp2p', 0, 0){
      unsigned int(32)[] compatible-brands;
}

class JP2PrefixBox() extends Box('jp2x'){
      int(8)[]    data;  // the data is the initial codestream part
}

class JP2SubSamplingBox () extends Box('jsub'){
      unsigned int(8) horizontal-sub;
      unsigned int(8) vertical-sub;
      unsigned    int(8) horizontal-offset;
      unsigned    int(8) vertical-offset;
}


      // Audio Sequences


class AudioSampleEntry() extends SampleEntry (AudioFormat){
      const unsigned int(32)[2] reserved = 0;
      unsigned int(16) channelcount;
      unsigned int(16) samplesize;
      unsigned int(16) pre-defined = 0;
      const unsigned int(16) reserved = 0 ;
      unsigned int(16) sampleratehi ; // typically is track timescale
      unsigned int(16) sampleratelo;
}
```

```
aligned(8) class SampleDescriptionBox (unsigned int(32) handler-type)
      extends FullBox('stsd', 0, 0){
      int i ;
      unsigned int(32) entry-count;
      for (i = 0 ; 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
                        HintSampleEntbry();
                        break;
            }
      }
}
```

6.2.16.2 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 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 .

`AudioFormat` is either 'raw ' or 'twos'.

`ChannelCount` is either 1 (mono) or 2 (stereo)

`SampleSize` is in bits, and takes the value 8 or 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

`Compressorname` may be set 0; it is a counted text string, padded to 32 bytes, for debugging purposes; for motion JPEG2000, the value "\017Motion JPEG2000" is recommended (\017 is 15, the length of the string as a byte)

`depth` takes one of the following values

```
0x18 – images are in color with no alpha
0x28 – images are in grayscale with no alpha
0x20 – images have alpha (gray or color)
```

`compatible-brands` is a list, filled to the end of the containing box, of JPEG2000 profiles, to which the associated codestreams conform; see Annex D for a provisional list;

`horizontal-sub` and `vertical-sub` indicate whether the chroma components of a YCbCr encoding were downsampled in the codestream; the value indicates the number of luminance samples to a single chroma sample in the given direction; This can assist decoders in memory allocation, and in using optimized sub-sampled display interfaces, for example.

`horizontal-offset` and `vertical-offset` specify the offset of the first chroma sample from the first luminance sample, as measured on the sample grid. If a CRG marker is present in the codestream, these values take precedence over those in the codestream. See the Annex “Indicating Sub-sampling Chroma Offset” for example values.

`fieldorder` describes the order of the two fields, and is only relevant if fieldcount equals 2:

0 field coding unknown

1 field with the topmost line is stored first in the sample; fields are in temporal order

6 field with the topmost line is stored second in the sample; fields are in temporal order

6.2.17 Sample Size Box

Box Type:	‘stsz’
Container:	Sample Table Box (‘stbl’)

Mandatory:	Yes
Quantity:	Exactly one

This box contains the sample count and a table giving the size 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. If the default size is indicated, then no table follows.

6.2.17.1 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=0; i < sample-count; i++) {
            unsigned int(32)  entry-size;
            }
      }
}
```

6.2.17.2 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.

`entry-count` is an integer that gives the number of entries in the following table

`entry-size` is integer specifying the size of a sample, indexed by its number.

6.2.18 Sample To Chunk Box

Box Type:	‘stsc’
Container:	Sample Table Box (‘stbl’)
Mandatory:	Yes
Quantity:	Exactly one

Samples within the media data are grouped into chunks. Chunks may be of different sizes, and the samples within a chunk may have different sizes. This table can be used to find a 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.

6.2.18.1 Syntax

```
aligned(8) class SampleToChunkBox
      extends FullBox(‘stsc’, version = 0, 0) {
      unsigned int(32)  entry-count;
      for (i=0; i < entry-count; i++) {
            unsigned int(32)  first-chunk;
            unsigned int(32)  samples-per-chunk;
            unsigned int(32)  sample-description-index;
      }
}
```

6.2.18.2 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 which share the same samples-per-chunk and sample-description-index

`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 which describes the samples in this chunk. The index ranges from 1 to the number of sample entries in the Sample Description Box

6.2.19 Chunk Offset Box

Box Type:	'stco', 'co64'
Container:	Sample Table Box ('stbl')
Mandatory:	Yes
Quantity:	Exactly one

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.

Note that 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 Motion JPEG2000 file with its meta-data (Movie Box) at the front, as the size of the Movie Box will affect the chunk offsets to the media data.

6.2.19.1 Syntax

```
aligned(8) class ChunkOffsetBox
      extends FullBox('stco', version = 0, 0) {
      unsigned int(32)  entry-count;
      for (i=0; 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=0; i < entry-count; i++) {
            unsigned int(64)  chunk-offset;
      }
}
```

6.2.19.2 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.

6.2.20 Free Space Box

Box Types:	'free', 'skip'
Container:	File
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. (Note that deleting the object may invalidate the offsets used in the sample table, unless this object is after all the media data).

6.2.20.1 Syntax

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

6.2.20.2 Semantics

`free-type` may be `'free'` or `'skip'`.

6.2.21 Edit Box

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.

Note that the Edit Box is optional. In the absence of this box, there is an implicit one-to-one mapping of these time-lines.

In the absence of an edit list, the presentation of a track starts immediately. An empty edit is used to offset the start time of a track.

6.2.21.1 Syntax

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

6.2.22 Edit List Box

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 may 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 may have been truncated if the edit enters it during its normal duration. If this is audio, that frame may need to be decoded, and then the final slicing done. Likewise, the duration of the last sample in an edit may 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

6.2.22.1 Syntax

```
aligned(8) class EditListBox extends FullBox('elst', version, 0) {
      unsigned int(32)  entry-count;
      for (i=0; 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;
      }
}
```

6.2.22.2 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 movie's time scale

`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 should never be an empty edit. Any difference between the movie's duration and the track's duration is expressed as an implicit empty edit.

`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 must contain the value 1.

6.2.23 User Data Box

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

This box contains objects which 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 draft. There may be multiple copyright boxes using different language codes.

6.2.23.1 Syntax

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

6.2.23.2 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 giving a copyright notice

6.2.24 Movie Extends Box

Box Type:	‘mvex’
Container:	Movie Box (‘moov’)
Mandatory:	No
Quantity:	Zero or one

This box warns readers that there may 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 the movie.

6.2.24.1 Syntax

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

6.2.25 Track Extends Box

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 may be saved in each Track Fragment Box.

A single flag is defined for a sample; it marks the sample as a difference (non-key) sample. This flag is not used in MJ2 as all JPEG2000 images are key images; there are no difference frames. The flag value is 0x10000.

6.2.25.1 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
}
```

6.2.25.2 Semantics

`track-id` identifies the track; this must correspond to a track in the movie
`default-` these fields set up defaults used in the track fragments.

6.2.26 Movie Fragment Box

Box Type:	‘moof’
Container:	File
Mandatory:	No
Quantity:	Zero or more

The movie fragments extend the movie 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. Note that the data reference index is in the sample description, so it is possible to build incremental movies where the media data is in files other than the movie file itself.

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.

6.2.26.1 Syntax

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

6.2.27 Movie Fragment Header Box

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 starts at 1 and increments by 1 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, or there are gaps in the sequence.

6.2.27.1 Syntax

```
aligned(8) class MovieFragmentHeaderBox
                  extends FullBox(‘mfhd’, 0, 0){
      unsigned int(32)  sequence-number;
}
```

6.2.27.2 Semantics

`sequence-number` the ordinal number of this fragment, in order, from 1 and up

6.2.28 Track Fragment Box

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.

6.2.28.1 Syntax

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

6.2.29 Track Fragment Header Box

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

Each movie fragment may add zero or more fragments to each track; and a track fragment may 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:

- 0x01 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 should all use the same data-reference (i.e., the data for these tracks should be in the same file).

- 0x02 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.
- 0x08 default-sample-duration-present
- 0x10 default-sample-size-present
- 0x20 default-sample-flags-present
- 0x100 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.

6.2.29.1 Syntax

```
aligned(8) class TrackFragmentHeaderBox
                  extends FullBox(‘tfhd’, 0, tf_flags){
      unsigned int(32)  track-ID;
      // all the following are optional fields
      signed 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
}
```

6.2.29.2 Semantics

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

6.2.30 Track Fragment Run Box

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 following flags are defined:

- 0x01 data-offset-present. 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,
- 0x04 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 should not be present.
- 0x08 sample-duration-present: indicates that each sample has its own duration, otherwise the default is used.
- 0x10 sample-size-present: each sample has its own size, otherwise the default is used.
- 0x20 sample-flags-present; each sample has its own flags, otherwise the default is used.
- 0x40 sample-composition-time-offsets-present; each sample has a composition time offset (used for I/P/B video in MPEG).

6.2.30.1 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 ]
}
```

6.2.30.2 Semantics

`sample-count` the number of samples being added in this fragment; also the number of rows in the following table (the rows may be empty)

7 Extensibility (Normative)

7.1 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 which together form a pair.

Commonly used in distributed computing are UUIDs (universal unique identifiers), which are 16 bytes. Any normative type specified here may 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 section 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 should 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 which have been written using the ‘uuid’ escape and the full UUID is not compliant; systems may choose to read objects using the uuid escape and an ISO uuid as equivalent to the box of the same type, or not.

7.2 Storage formats

The main file containing the meta-data 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 meta-data declaration set in it, that meta-data is not part of the overall presentation. This allows small presentation files to be aggregated into a larger overall presentation by building new meta-data 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.

Annex A: Overview and Introduction (Informative)

This section provides an introduction to the file format, which may assist readers in understanding the overall concepts underlying the file format. It forms an informative annex to this specification.

A.1 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 which 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 which 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 **atom**s in the other versions of this file format). There is no data outside the box structure. All the meta-data, 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 meta-data. The media data may be in the same file (contained in one or more boxes), or can be in other files; the meta-data permits referring to other files by means of URLs. The placement of the media data within these secondary files is entirely described by the meta-data 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 which 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.2 Physical structure of the media

The boxes which 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, even in a file where the media data for several tracks has been interleaved, 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.3 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.4 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.5 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).

A.6 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. This algorithm is general, but note for example that every frame is a sync point in Motion JPEG2000.

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) Knowing which chunk contained the sample in question, use the Chunk Offset Box to figure out where that chunk begins.

4) 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.7 Fragmented movie files

This section introduces a technique which may be used in MJ2 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 meta-data (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 which have non-default values need include those values. This makes the common case — regular, repeating, structures — compact, without disabling the incremental building of movies which 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 which doesn't understand fragments will play just this initial movie. Software which 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: Guidelines for use of the JPEG2000 Codec (Informative)

B.1 Introduction

Certain artifacts have been observed when using the 15444-1 codec to create motion sequences. Experiments have been performed to identify the cause of these artifacts and indicate encoding strategies that would be likely to mitigate them. This Annex is intended to review the results of the experiments that were performed and recommend possible methods that may be employed. These methods are not necessarily the best or only approach to addressing these observed artifacts. This Annex is intended as a point of reference for individuals implementing 15444-3 (Motion JPEG2000).

Section B.2 suggests particular frequency weighting for motion sequences. Section B.3 addresses sub-sampling issues.

B.2 Frequency Weighting for Motion Sequences

B.2.1 Background

The objective of employing frequency weighting to motion images is to explore methods that might improve subjective quality. Tuning of weighting values for chroma component is an important area to investigate. An experiment (WG1N1689) showed that there are some flickering artifacts when displaying a moving sequence of images although individual pictures do not show an annoying artifact. The frequency weighting was observed to reduce this artifacts in all test sequences, especially at low to middle bitrate (for example 0.25 to 0.50bpp for CIF). A finer and better adjustment of visual weights to improve the image sequence's quality may provide additional improvements.

For color images the frequency weighting tables of the Y, Cr, and Cb components should differ in order to take advantage of the properties of the human visual system. For example, it is usually desirable to emphasize the luminance component more than the chrominance components.

B.2.2 Recommended frequency weighting tables

Table 3 specifies a set of CSF weights which were designed for the luminance component based on the CSF value at the mid-frequency of each subband. The other two tables (Table 5 and Table 7) specifies sets of CSF weights for Cb and Cr. Note that the tables are intended for a 5-level wavelet decomposition.

The table does not include the weight for the lowest frequency subband, nLL, which is always 1. Levels 1, 2, …,5 denote the subband levels in low to high frequency order. (HL, LH, HH) denotes the three frequency orientations within each subband.

Table 3: CSF weights for Luminance

Level	HL	LH	HH
1	1.000000	1.000000	1.000000
2	1.000000	1.000000	1.000000
3	0.999994	0.999994	0.999988
4	0.837755	0.837755	0.701837
5	0.275783	0.275783	0.090078

Table 5: CSF weights for Cb

Level	HL	LH	HH
1	0.812612	0.812612	0.737656
2	0.679829	0.679829	0.567414
3	0.488887	0.488887	0.348719
4	0.267216	0.267216	0.141965
5	0.089950	0.089950	0.027441

Table 7: CSF weights for Cr

Level	HL	LH	HH
1	0.856065	0.856065	0.796593
2	0.749805	0.749805	0.655884
3	0.587213	0.587213	0.457826
4	0.375176	0.375176	0.236030
5	0.166647	0.166647	0.070185

B.3 Encoder sub-sampling of components

In the most of all cases, the format of motion sequences is either YcbCr (BT-601-5 for standard TV and BT-709-4 for HDTV) or YUV. In these, color components are already sub-sampled and the sub-sampled signal may be the input of the Motion-JPEG2000 encoder. Although the encoder sub-sampling of color components is not recommended in 15444-1, the sub-sampling may be commonly used in Motion-JPEG2000. Therefore, CSF weights also should be weighted directly to sub-sampled color components.

Sub-sampling of chroma components in JPEG2000 can be achieved in two ways: (a) the highest-frequency sub-bands may be omitted, and (b) if no ICT is used, then the codestream permits the components to differ in size.

B.4 References

[1] T.Fukuhara and S.Kimura, “Results of MJ2 Core Experiment No.1 (Visual Weighting)”, ISO/IEC/JTC1 SC29/WG1/N1689, May, 2000

[2] Troy Chinen, Marcus Nadenau, “Report on CE C03 (Optimizing Color Image Compression)”, ISO/IEC/JTC1 SC29/WG1/N1587, May, 2000

Annex C: File and Codestream profiles (Normative)

This Annex normatively specifies the ways in which Motion JPEG2000 files may be profiled.

There are three areas for which profiles and levels can be declared:

a) The optional still image which may occur at the top-level of a Motion JPEG2000 file.
b) The characteristics of the individual frames (codestreams) which form a motion sequence.
c) The overall characteristics of the Motion Presentation itself.

The profile for the optional still image is declared by including the correct 'compatible brands' in the top-level File Type Box. These profiles (brands) are specified for JP2 files, and that specification should be followed here.

Profile code-points should be used to profile the images that form the frames of motion sequences. These code-points are placed within the JP2 Profile Box within the sample description.

The characteristics of the motion presentation are declared by placing the brands of the compatible profiles in the top-level file-type box. This specification defines two profiles:

1) Unrestricted. The brand 'mjp2' indicates unrestricted conformance to this specification and should be placed in the top-level File Type Box in all files.
2) Simple. The brand 'mj2s' indicates the simple Motion JPEG2000 profile, as defined below.

C.1 Motion JPEG2000 Simple Profile

File conforming to the simple profile have the following characteristics:

1) Exactly one video track is present.
2) At most a single audio track, using only 8 or 16-bit raw audio, is present.
3) Each track should have exactly one sample description, used by all samples.
4) The sample rate of the audio, if present, may not exceed 48 kHz.
5) The frame rate of the video shall not exceed 30 frames per second.
6) The video codestream profile level shall be level 0 for both the motion sequence and the still image, if present.
7) The file is self-contained; no data references are used, and therefore all media data is contained within the single file.
8) The media data in the Media Data Box(es) is placed within the box(es) in temporal order.
9) If more than one track is present, the media data for the tracks is interleaved, with a granularity no greater than the greater of (a) the duration of a single 'sample' (in file format terms) or (b) one second.
10) The transformation matrices used are restricted to uniform scaling and 90º rotation.

Annex D: Indicating Sub-sampling Chroma Offset (Informative)

This annex provides informative material for the Sub-sampling Box 'jsub' (see above, section **Error! Reference source not found.**), with offset values for common sub-sampling formats.

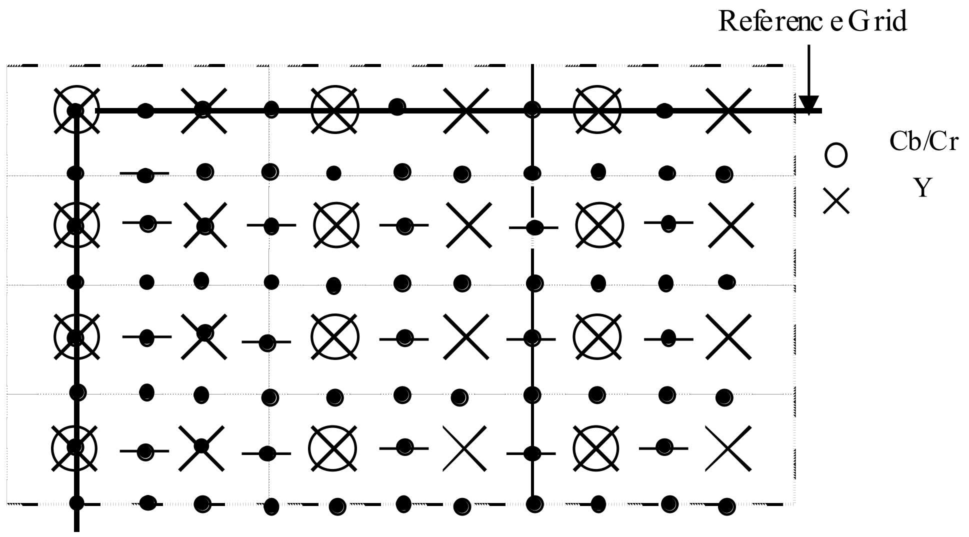

Figure 1 4:2:2 format (CCIR601,H.262,MPEG-2,MPEG-4,Exif(recommended))

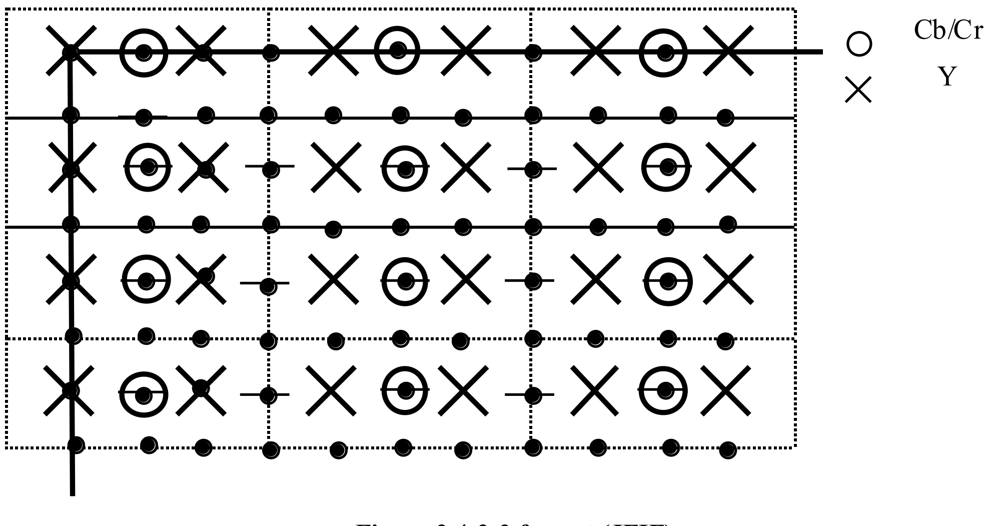

Figure 2 4:2:2 format (JFIF)

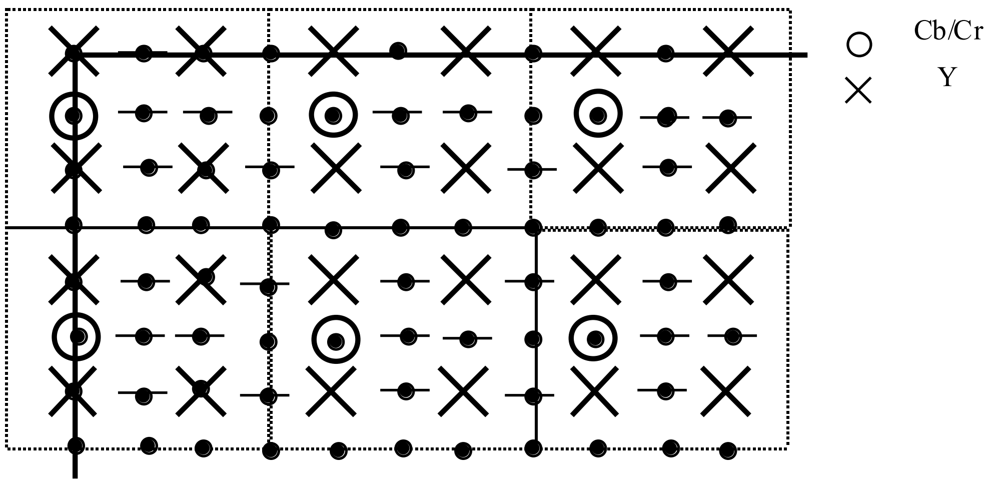

Figure 3 4:2:0 format (H.262,MPEG-2,MPEG-4)

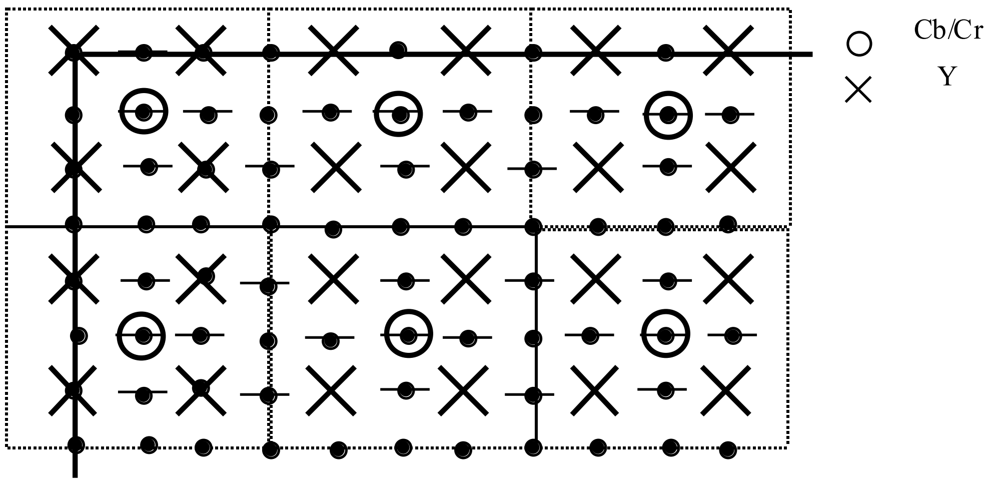

Figure 4 4:2:0 format (MPEG-1,H.261,JFIF,Exif(recommended))

	Figure 1	Figure 2	Figure 3	Figure 4
Horizontal-offset	0	1	0	1
Vertical-offset	0	0	1	1

Table 9 Chroma Phase Values