INTERNATIONAL STANDARD

ISO/IEC 15444-12

Third edition
2008-10-15

AMENDMENT 1
2009-11-15

Information technology — JPEG 2000 image coding system —

Part 12: ISO base media file format

AMENDMENT 1: General improvements including hint tracks, metadata support and sample groups

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

Partie 12: Format ISO de base pour les fichiers médias

AMENDEMENT 1: Améliorations générales, y compris «hint tracks», support de métadonnées et groupes d'échantillons

Reference number
ISO/IEC 15444-12:2008/Amd.1:2009(E)

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

Published in Switzerland

Foreword

ISO (the International Organization for Standardization) and IEC (the International Electrotechnical Commission) form the specialized system for worldwide standardization. National bodies that are members of ISO or IEC participate in the development of International Standards through technical committees established by the respective organization to deal with particular fields of technical activity. ISO and IEC technical committees collaborate in fields of mutual interest. Other international organizations, governmental and non-governmental, in liaison with ISO and IEC, also take part in the work. In the field of information technology, ISO and IEC have established a joint technical committee, ISO/IEC JTC 1.

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

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

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

Amendment 1 to ISO/IEC 15444-12:2008 was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee SC 29, *Coding of audio, picture, multimedia and hypermedia information*.

This Amendment provides a number of new hint track formats and improvements in other areas, including metadata support.

Information technology — JPEG 2000 image coding system —

Part 12:
ISO base media file format

AMENDMENT 1: General improvements including hint tracks, metadata support and sample groups

Reformat the table of contents, including the new subclauses and revised page numbers.

Add the following normative references to Clause 2:

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

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

In 6.2.3, Table 1, add the following line after 'tref':

		trgr					8.3.4	*track grouping indication*

In 6.2.3, Table 1, add the following line after 'ctts':

					cslg		8.6.1.4	*composition to decode timeline mapping*

In 6.2.3, add the following after 'paen':

			fire				8.13.7	*file reservoir*

In 6.2.3, Table 1, add the following lines after 'tsel':

	idat						8.11.11	*item data*
	iref						8.11.12	*item reference*

After 6.2.3, add the following new subclause:

6.2.4 URIs as type indicators

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

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

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

Replace Clause 7 with the following:

7 Streaming Support

7.1 Handling of Streaming Protocols

The file format supports streaming of media data over a network as well as local playback. The process of sending protocol data units is time-based, just like the display of time-based data, and is therefore suitably described by a time-based format. A file or 'movie' that supports streaming includes information about the data units to stream. This information is included in additional tracks of the file called "hint" tracks. Hint tracks may also be used to record a stream; these are called Reception Hint Tracks, to differentiate them from plain (or server, or transmission) hint tracks.

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

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

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

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

NOTE 1: Players may reproduce the packet stream that was received based on the reception hint tracks and process the reproduced packet stream as if it was newly received.

7.2 Protocol 'hint' tracks

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

- The media data is represented as a set of network-independent standard tracks, which may be played, edited, and so on, as normal;

- There is a common declaration and base structure for hint tracks; this common format is protocol independent, but contains the declarations of which protocol(s) are described in the hint track(s);
- There is a specific design of the hint tracks for each protocol that may be transmitted; all these designs use the same basic structure. For example, there may be designs for RTP (for the Internet) and MPEG-2 transport (for broadcast), or for new standard or vendor-specific protocols.

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

7.3 Hint Track Format

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

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

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

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

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

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

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

Hint tracks may be created by an authoring tool, or may be added to an existing presentation by a hinting tool. Such a tool serves as a ‘bridge’ between the media and the protocol, since it intimately understands both. This

permits authoring tools to understand the media format, but not protocols, and for servers to understand protocols (and their hint tracks) but not the details of media data.

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

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

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

In 8.3.2.1, change:

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

to:

Server hint tracks should have the `track_in_movie` and `track_in_preview` track header flags set to 0, so that they are ignored for local playback and preview.

In 8.3.3.3 (Track Reference Box, Semantics), replace the definition of reference_type as follows:

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

- 'hint' the referenced track(s) contain the original media for this hint track
- 'cdsc' this track describes the referenced track.
- 'hind' this track depends on the referenced hint track, i.e., it should only be used if the referenced hint track is used.
- '`vdep`'this track contains auxiliary depth video information for the referenced video track
- '`vplx`'this track contains auxiliary parallax video information for the referenced video track

After 8.3.3, add the following new subclause:

8.3.4 Track Group Box

8.3.4.1 Definition

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

This box enables indication of groups of tracks, where each group shares a particular characteristic or the tracks within a group have a particular relationship. The box contains zero or more boxes, and the particular

characteristic or the relationship is indicated by the box type of the contained boxes. The contained boxes include an identifier, which can be used to conclude the tracks belonging to the same track group. The tracks that contain the same type of a contained box within the Track Group Box and have the same identifier value within these contained boxes belong to the same track group.

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

8.3.4.2 Syntax

```
aligned(8) class TrackGroupBox('trgr') {
}

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

8.3.4.3 Semantics

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

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

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

In 8.4.3.1, (Handler Reference Box, definition), before the NOTEs, add the following paragraph:

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

In 8.4.3.3, after the line for ‘meta’ add the following line:

`‘auxv’` Auxiliary Video track

At the end of 8.5.2.1 (Sample Description, Definition), add the following:

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

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

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

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

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

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

In 8.5.2.2, add before "// Visual Sequences":

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

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

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

At the end of 8.5.2.3, add the following:

`theURI` is a URI formatted according to the rules in 6.2.4;
`uri_initialization_data` is opaque data whose form is defined in the documentation of the URI form.

In 8.6.1.3.1 (Composition Time to Sample, Definition), replace the body text with the following:

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

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

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

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

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

Hint tracks do not use this box.

Move the table and its header 'For example in Table 2' from 8.6.1.3.2 to the end of 8.6.1.3.1

Replace 8.6.1.3.2 with the following:

8.6.1.3.2 Syntax

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

In 8.6.1.3.3, replace the following line with this revised definition:

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

After 8.6.1.3, add the following:

8.6.1.4 Composition to Decode Box

8.6.1.4.1 Definition

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

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

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

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

8.6.1.4.2 Syntax

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

8.6.1.4.3 Semantics

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

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

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

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

`compositionEndTime`: the composition time plus the composition duration, of the sample with the largest computed composition time (CTS) in the media of this track

In 8.6.4.1 (Independent and Disposable Samples, Definition), add the following paragraph before the penultimate paragraph (starting "The size of the table"):

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

Replace 8.6.4.2 with the following:

8.6.4.2 Syntax

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

Add the following to the beginning of 8.6.4.3:

`is_leading` takes one of the following four values:

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

In 8.8.3.1 (Track Extends, definition), replace the code and the following paragraph, with the following:

```
        bit(4)    reserved=0;
        unsigned int(2) is_leading;
        unsigned int(2) sample_depends_on;
        unsigned int(2) sample_is_depended_on;
        unsigned int(2) sample_has_redundancy;
        bit(3)    sample_padding_value;
        bit(1)    sample_is_difference_sample;
              // i.e. when 1 signals a non-key or non-sync sample
        unsigned int(16)  sample_degradation_priority;
```

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

In 8.9.1 (Sample Group Structures, Introduction), add the following before the final paragraph (that starts "One example"):

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

Add the following to the end of 8.9.2.1 (Sample to Group, Definition):

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

Replace 8.9.2.2 as follows:

8.9.2.2 Syntax

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

Replace the first two paragraphs (preceding "entry_count") of 8.9.2.3 with the following:

`version` is an integer that specifies the version of this box, either 0 or 1.

`grouping_type` is an integer that identifies the type (i.e. criterion used to form the sample groups) of the sample grouping and links it to its sample group description table with the same value for grouping type. At most one occurrence of this box with the same value for `grouping_type` (and, if used, `grouping_type_parameter`) shall exist for a track.

`grouping_type_parameter` is an indication of the sub-type of the grouping

Replace 8.11.1.2 and 8.11.1.3 (Meta Box, Syntax and Semantics) with the following:

8.11.1.2 Syntax

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

8.11.1.3 Semantics

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

Replace 8.11.3 with the following:

8.11.3 The Item Location Box

8.11.3.1 Definition

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

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

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

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

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

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

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

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

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

- If the offset is not identified (the field has a length of zero), then the beginning of the source (offset 0 into the file, idat box, or other item) is implied.

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

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

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

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

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

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

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

8.11.3.2 Syntax

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