INTERNATIONAL STANDARD

ISO/IEC 14496-12

Fifth edition
2015-12-15

AMENDMENT 2
2018-02

Information technology — Coding of audio-visual objects —

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

AMENDMENT 2: Support for image file format

Technologies de l'information — Codage des objets audiovisuels —

Partie 12: Format ISO de base pour les fichiers médias

AMENDEMENT 2: Support pour fichiers au format image

Reference number
ISO/IEC 14496-12:2015/Amd.2:2018(E)

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.

The procedures used to develop this document and those intended for its further maintenance are described in the ISO/IEC Directives, Part 1. In particular the different approval criteria needed for the different types of ISO documents should be noted. This document was drafted in accordance with the editorial rules of the ISO/IEC Directives, Part 2 (see www.iso.org/directives).

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. ISO shall not be held responsible for identifying any or all such patent rights. Details of any patent rights identified during the development of the document will be in the Introduction and/or on the ISO list of patent declarations received (see www.iso.org/patents).

Any trade name used in this document is information given for the convenience of users and does not constitute an endorsement.

For an explanation on the voluntary nature of standards, the meaning of ISO specific terms and expressions related to conformity assessment, as well as information about ISO's adherence to the World Trade Organization (WTO) principles in the Technical Barriers to Trade (TBT) see the following URL: www.iso.org/iso/foreword.html.

This document was prepared by ISO/IEC JTC 1, *Information technology*, SC 29, *Coding of audio, picture, multimedia and hypermedia information*.

A list of all parts in the ISO 14496 series can be found on the ISO website.

Information technology — Coding of audio-visual objects —

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

AMENDMENT 2: Support for image file format

3.1

Add term entry and renumber accordingly so that all terms and definitions appear in alphabetical order:

3.1.1

thumbnail image

smaller-resolution representation of an image

8.3.3.3

Add the following additional `reference_type` values for `TrackReferenceBox` in 8.3.3.3:

'`thmb`': this track contains thumbnail images for the referenced track. A thumbnail track shall not be linked to another thumbnail track with the '`thmb`' item reference.

'`auxl`': this track contains auxiliary media for the indicated track (e.g. depth map or alpha plane for video).

NOTE 1 A track with reference type '`auxl`' might have a coding dependency; its use is clarified by specifications that use it.

NOTE 2 When multiple track references would describe an auxiliary video track, derived specifications might constrain or recommend which track references are used. For example, derived specifications might constrain or recommend whether to use '`vdep`' or '`auxl`' or both for auxiliary depth video track.

8.6.6.2

Replace

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

with

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

8.6.6.3

Add after the definition of "version":

> `flags` specifies repetition of the edit list as follows. (`flags` & 1) equal to 0 specifies that the edit list is not repeated, while (`flags` & 1) equal to 1 specifies that the edit list is repeated. The values of `flags` greater than 1 are reserved. When an `EditListBox` indicates the playback of zero or one samples, (`flags` & 1) shall be equal to 0.

NOTE When the edit list is repeated, media at time 0 resulting from the edit list follows immediately the media having the largest time resulting from the edit list. In other words, the edit list is repeated seamlessly.

8.11.6.1

Add to the end of 8.11.6.1:

The flags field of ItemInfoEntry with version greater than or equal to 2 is specified as follows:

(flags & 1) equal to 1 indicates that the item is not intended to be a part of the presentation,
(flags & 1) equal to 0 indicates that the item is intended to be a part of the presentation.

8.11.6.2

Replace

```
aligned(8) class ItemInfoEntry
   extends FullBox('infe', version, 0) {
```

with

```
aligned(8) class ItemInfoEntry
   extends FullBox('infe', version, flags) {
```

8.11.14

Add the following as 8.11.14, renumbering as needed:

8.11.14 Item Properties Box

8.11.14.1 Definition

Box Type: 'iprp'

Container: MetaBox ('meta')

Mandatory: No

Quantity: Zero or one

The ItemPropertiesBox enables the association of any item with an ordered set of item properties. Item properties are small data records.

The ItemPropertiesBox consists of two parts: ItemPropertyContainerBox that contains an implicitly indexed list of item properties, and one or more ItemPropertyAssociationBox(es) that associate items with item properties.

Each item property is a Box or FullBox. The boxtype of the item property specifies the property type. The FreeSpaceBox may occur in the ItemPropertyContainerBox; it has no meaning, and should not be associated with any item.

Each property association may be marked as either essential or non-essential. A reader shall not process an item that is associated with a property that is not recognized or not supported by the reader and that is marked as essential to the item. A reader may ignore an associated item property that is marked non-essential to the item.

Specifications deriving from this specification may specify property types and the respective item property box definitions as well as constraints and requirements for the property associations.

When defining item properties, it is recommended that they be small. When large data records need to be associated with an item, a separate item and item reference are more suitable.

Each ItemPropertyAssociationBox shall be ordered by increasing item_ID, and there shall be at most one occurrence of a given item_ID, in the set of ItemPropertyAssociationBox boxes. The version 0 should be used unless 32-bit item_ID values are needed; similarly, flags should be equal to 0 unless there are more than 127 properties in the ItemPropertyContainerBox. There shall be at most one ItemPropertyAssociationBox with a given pair of values of version and flags.

8.11.14.2 Syntax

```
aligned(8) class ItemProperty(property_type)
   extends Box(property_type)
{
}


aligned(8) class ItemFullProperty(property_type, version, flags)
   extends FullBox(property_type, version, flags)
{
}


aligned(8) class ItemPropertyContainerBox
   extends Box('ipco')
{
   Box properties[];   // boxes derived from
      // ItemProperty or ItemFullProperty or FreeSpaceBox(es)

      // to fill the box
}

aligned(8) class ItemPropertyAssociationBox
   extends FullBox('ipma', version, flags)


{
   unsigned int(32) entry_count;
   for(i = 0; i < entry_count; i++) {
      if (version < 1)
         unsigned int(16)   item_ID;
      else
         unsigned int(32)   item_ID;
      unsigned int(8) association_count;
      for (i=0; i<association_count; i++) {
         bit(1) essential;
         if (flags & 1)
            unsigned int(15) property_index;
         else
            unsigned int(7) property_index;
      }
   }
}


aligned(8) class ItemPropertiesBox
      extends Box('iprp') {
   ItemPropertyContainerBox property_container;
   ItemPropertyAssociationBox association[];
}
```

8.11.14.3 Semantics

`item_ID` identifies the item with which properties are associated

`essential` when set to 1 indicates that the associated property is essential to the item, otherwise it is non-essential

`property_index` is either 0 indicating that no property is associated (the essential indicator shall also be 0), or is the 1-based index (counting all boxes, including `FreeSpace` boxes) of the associated property box in the `ItemPropertyContainerBox` contained in the same `ItemPropertiesBox`.

8.18

Add the following as 8.18, renumbering as needed:

8.18 Entity grouping

8.18.1 General

An entity group is a grouping of items, which may also group tracks. The entities in an entity group share a particular characteristic or have a particular relationship, as indicated by the grouping type.

Entity groups are indicated in `GroupsListBox`. Entity groups specified in `GroupsListBox` of a file-level `MetaBox` refer to tracks or file-level items. Entity groups specified in `GroupsListBox` of a movie-level `MetaBox` refer to movie-level items. Entity groups specified in `GroupsListBox` of a track-level `MetaBox` refer to track-level items of that track.

`GroupsListBox` contains `EntityToGroupBoxes`, each specifying one entity group.

8.18.2 Groups List box

8.18.2.1 Definition

Box Type: '`grpl`'

Container: `MetaBox` that is not contained in `AdditionalMetadataContainerBox`

Mandatory: No

Quantity: Zero or One

The `GroupsListBox` includes the entity groups specified for the file. This box contains a set of full boxes, each called an `EntityToGroupBox`, with four-character codes denoting a defined grouping type.

The `GroupsListBox` shall not be present in `AdditionalMetadataContainerBox`.

When `GroupsListBox` is present in a file-level `MetaBox`, there shall be no `item_ID` value in `ItemInfoBox` in any file-level `MetaBox` that is equal to the `track_ID` value in any `TrackHeaderBox`.

8.18.2.2 Syntax

```
aligned(8) class GroupsListBox extends Box('grpl') {
}
```

8.18.3 Entity to Group box

8.18.3.1 Definition

Box Type: As specified below with the grouping_type value for the EntityToGroupBox

Container: GroupsListBox

Mandatory: No

Quantity: One or more

The EntityToGroupBox specifies an entity group.

The box type (grouping_type) indicates the grouping type of the entity group. Each grouping_type code is associated with semantics that describe the grouping. The following grouping_type value is specified:

> 'altr': The items and tracks mapped to this grouping are alternatives to each other, and only one of them should be played (when the mapped items and tracks are part of the presentation, e.g. are displayable items or tracks) or processed by other means (when the mapped items or tracks are not part of the presentation, e.g. are metadata). A player should select the first entity from the list of entity_id values that it can process (e.g. decode and play for mapped items and tracks that are part of the presentation) and that suits the application needs. Any entity_id value shall be mapped to only one grouping of type 'altr'. An alternate group of entities consists of those items and tracks that are mapped to the same entity group of type 'altr'.

NOTE EntityToGroupBox could have grouping_type specific extensions.

8.18.3.2 Syntax

```
aligned(8) class EntityToGroupBox(grouping_type, version, flags)
extends FullBox(grouping_type, version, flags) {
   unsigned int(32) group_id;
   unsigned int(32) num_entities_in_group;
   for(i=0; i<num_entities_in_group; i++)
      unsigned int(32) entity_id;
// the remaining data may be specified for a particular grouping_type
}
```

8.18.3.3 Semantics

group_id is a non-negative integer assigned to the particular grouping that shall not be equal to any group_id value of any other EntityToGroupBox, any item_ID value of the hierarchy level (file, movie or track) that contains the GroupsListBox, or any track_ID value (when the GroupsListBox is contained in the file level).

num_entities_in_group specifies the number of entity_id values mapped to this entity group.

entity_id is resolved to an item, when an item with item_ID equal to entity_id is present in the hierarchy level (file, movie or track) that contains the GroupsListBox, or to a track, when a track with track_ID equal to entity_id is present and the GroupsListBox is contained in the file level.

10.7

Add the following as 10.7, renumbering as needed:

10.7 Sample-to-item sample grouping

10.7.1 Definition

Samples of a track can be linked to one more metadata items using the sample-to-item sample grouping. The MetaBox containing the referred items is resolved as specified in the semantics below.

The sample-to-item sample grouping is allowed for any types of tracks, and its syntax and semantics are unchanged regardless of the track handler type.

In the absence of this sample group, the entire track-level MetaBox, if any, is applicable to every sample.

10.7.2 Syntax

```
class SampleToMetadataItemEntry()
extends SampleGroupDescriptionEntry('stmi') {
   unsigned int(32) meta_box_handler_type;
   unsigned int(32) num_items;
   for(i = 0; i < num_items; i++) {
      unsigned int(32) item_id[i];
   }
}
```

10.7.3 Semantics

meta_box_handler_type informs about the type of metadata schema used by the MetaBox which is referenced by the items in this sample group. When there are multiple MetaBoxes with the same handler types, the MetaBox referred to in this sample group entry is the first MetaBox fulfilling one of the following ordered constraints:

— a MetaBox included in the current track, contained in AdditionalMetadataContainerBox, and with handler_type equal to meta_box_handler_type;

— a MetaBox included in the current track, contained in AdditionalMetadataContainerBox, and with handler_type equal to meta_box_handler_type;

— a MetaBox included in MovieBox, not contained in AdditionalMetadataContainerBox, and with handler_type equal to meta_box_handler_type;

— a MetaBox included in MovieBox, not contained in AdditionalMetadataContainerBox, and with handler_type equal to meta_box_handler_type;

— a MetaBox included in the root level of the file, not contained in AdditionalMetadataContainerBox, and with handler_type equal to meta_box_handler_type

— a MetaBox included in the root level of the file, not contained in AdditionalMetadataContainerBox, and with handler_type equal to meta_box_handler_type `num_items` counts the number of items referenced by this sample group.

`item_id[i]` specifies the `item_ID` value of an item that applies to or is valid for the sample mapped to this sample group description entry.

10.8

Add the following as 10.8, renumbering as needed:

10.8 Dependent random access point (DRAP) sample grouping

10.8.1 Definition

A dependent random access point (DRAP) sample is a sample after which all samples in decoding order can be correctly decoded if the closest initial sample preceding the DRAP sample is available for reference. The initial sample is a SAP sample of SAP type 1, 2 or 3 that is marked as such either by being a Sync sample or by the SAP sample group. For example, if the 32nd sample in a file is an initial sample consisting of an I-picture, the 48th sample may consist of a P-picture and be marked as a member of the dependent random access point sample group, thereby indicating that random access can be performed at the 48th sample by first decoding the 32nd sample (ignoring samples 33–47) and then continuing to decode from the 48th sample.

A sample can be a member of the dependent random access point `sample group` (and hence called a DRAP sample) only if the following conditions are true.

— The DRAP sample references only the closest preceding initial sample.

— The DRAP sample and all samples following the DRAP sample in output order can be correctly decoded when starting decoding at the DRAP sample after having decoded the closest preceding SAP sample of type 1, 2 or 3 marked as a Sync sample or by the SAP sample group.

NOTE DRAP samples can only be used in combination with SAP samples of type 1, 2 and 3. This is in order to enable the functionality of creating a decodable sequence of samples by concatenating the preceding SAP sample with the DRAP sample and the samples following the DRAP sample in output order.

10.8.2 Syntax

```
class VisualDRAPEntry()
extends VisualSampleGroupEntry('drap') {
   unsigned int(3) DRAP_type;
   unsigned int(29) reserved = 0;
}
```

10.8.3 Semantics

`DRAP_type` is a non-negative integer. When `DRAP_type` is in the range of 1 to 3 it indicates the `SAP_type` (as specified in Annex I) that the DRAP sample would have corresponded to, had it not depended on the closest preceding SAP. Other type values are reserved.

`reserved` shall be equal to 0. The semantics of this clause only apply to sample group description entries with `reserved` equal to 0. Parsers shall allow and ignore sample group description entries with `reserved` greater than 0 when parsing this sample group.