INTERNATIONAL STANDARD

ISO/IEC 15444-12

Second edition
2005-04-01

Corrected version
2005-10-01

AMENDMENT 2
2008-02-01

Information technology — JPEG 2000 image coding system —

Part 12: ISO base media file format

AMENDMENT 2: Hint track format for ALC/LCT and FLUTE transmission and multiple meta box support

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

Partie 12: Format ISO de base pour les fichiers médias

AMENDEMENT 2: Format de piste optimisé pour transmission ALC/LCT et FLUTE et support de boîte "meta" multiple

Reference number
ISO/IEC 15444-12:2005/Amd.2:2008(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 2 to ISO/IEC 15444-12:2005 was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee SC 29, *Coding of audio, picture, multimedia and hypermedia information*.

Information technology — JPEG 2000 image coding system —

Part 12: ISO base media file format

AMENDMENT 2: Hint track format for ALC/LCT and FLUTE transmission and multiple meta box support

Replace Clause 3 with the following:

3 Terms, definitions and abbreviated terms

For the purposes of this document, the following terms and definitions and abbreviated terms apply.

3.1 Terms and definitions

3.1.1
Box
object-oriented building block defined by a unique type identifier and length (called 'atom' in some specifications, including the first definition of MP4)

3.1.2
Chunk
contiguous set of samples for one track

3.1.3
Container Box
box whose sole purpose is to contain and group a set of related boxes

3.1.4
Hint Track
special track which does not contain media data. Instead it contains instructions for packaging one or more tracks into a streaming channel

3.1.5
Hinter
tool that is run on a file containing only media, to add one or more hint tracks to the file and so facilitate streaming

3.1.6
Movie Box
container box whose sub-boxes define the metadata for a presentation ('moov')

3.1.7
Media Data Box
container box which can hold the actual media data for a presentation ('mdat')

3.1.8
ISO Base Media File
name of the file format described in this specification

3.1.9
Presentation
one or more motion sequences (q.v.), possibly combined with audio

3.1.10
Sample
In non-hint tracks, a sample is an individual frame of video, a series of video frames in decoding order, or a compressed section of audio in decoding order. In hint tracks, a sample defines the formation of one or more streaming packets. No two samples within a track may share the same time-stamp.

3.1.11
Sample Description
structure which defines and describes the format of some number of samples in a track

3.1.12
Sample Table
packed directory for the timing and physical layout of the samples in a track

3.1.13
Track
Collection of related samples (q.v.) in an ISO base media file. For media data, a track corresponds to a sequence of images or sampled audio. For hint tracks, a track corresponds to a streaming channel.

3.2 Abbreviated terms

ALC	Asynchronous Layered Coding
FD	File Delivery
FDT	File Delivery Table
FEC	Forward Error Correction
FLUTE	File Delivery over Unidirectional Transport
IANA	Internet Assigned Numbers Authority
LCT	Layered Coding Transport
MBMS	Multimedia Broadcast/Multicast Service

In 4.2 Object Structure, after "The fields in the objects are stored with the most significant byte first, commonly known as network byte order or big-endian format.", insert the following:

When fields smaller than a byte are defined, or fields span a byte boundary, the bits are assigned from the most significant bits in each byte to the least significant. For example, a field of two bits followed by a field of six bits has the two bits in the high order bits of the byte.

In 4.3.1, add the following at the end of the subclause:

All file format brands defined in this specification are included in Annex E with a summary of which features they require.

In 6.2.3, add the following entries at the end of Table 1 (correctly cross-referenced):

	fiin						8.46.2	*file delivery item information*
		paen					8.46.2	*partition entry*
			fpar				8.46.3	*file partition*
			fecr				8.46.4	*FEC reservoir*
		segr					8.46.5	*file delivery session group*
		gitn					8.46.6	*group id to name*
		tsel					8.48.2	*track selection*
meco							8.44.9	*additional metadata container*
	mere						8.44.10	*metabox relation*

In 8.6.3, add the following reference_type to the end of the list:

- 'hind' this track depends on the referenced hint track, i.e., it should only be used if the referenced hint track is used.

In 8.9.3, replace:

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

with:

`handler_type` when present in a meta box, contains an appropriate value to indicate the format of the meta box contents. The value `'null'` can be used in the primary meta box to indicate that it is merely being used to hold resources.

In 8.44.1.1, replace:

"Box type: … Zero or one", i.e. the first four lines at the beginning,

with:

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

In 8.44.1.1 Definition, replace:

At most one meta box may occur at each of the file level, movie level, or track level.

with:

At most one meta box may occur at each of the file level, movie level, or track level, unless they are contained in an additional metadata container box ('meco').

In 8.44.6.1, insert the following text at the end of the subclause:

Two versions of the item info entry are defined. Version 1 includes additional information to version 0 as specified by an extension type. For instance, it shall be used with extension type 'fdel' for items that are referenced by the file partition box ('fpar'), which is defined for source file partitionings and applies to file delivery transmissions.

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

In 8.44.6.2, replace the entire text with the following:

```
aligned(8) class ItemInfoExtension(unsigned int(32) extension_type)
{
}

aligned(8) class FDItemInfoExtension() extends ItemInfoExtension (’fdel’)
{
   string            content_location;
   string            content_MD5;
   unsigned int(64)  content_length;
   unsigned int(64)  transfer_length;
   unsigned int(8)   entry_count;
   for (i=1; i <= entry_count; i++)
      unsigned int(32)  group_id;
}

aligned(8) class ItemInfoEntry
      extends FullBox(‘infe’, version, 0) {
   if ((version == 0) || (version == 1)) {
      unsigned int(16)  item_ID;
      unsigned int(16)  item_protection_index
      string            item_name;
      string            content_type;
      string            content_encoding;   //optional
   }
   if (version == 1) {
      unsigned int(32)  extension_type;     //optional
      ItemInfoExtension(extension_type);    //optional
   }
}

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

In 8.44.6.3, replace the entire text with the following:

`item_id` contains either 0 for the primary resource (e.g., the XML contained in an ‘xml ‘ box) or the ID of the item for which the following information is defined.

`item_protection_index` contains either 0 for an unprotected item, or the one-based index into the item protection box defining the protection applied to this item (the first box in the item protection box has the index 1).

`item_name` is a null-terminated string in UTF-8 characters containing a symbolic name of the item (source file for file delivery transmissions).

`content_type` is a null-terminated string in UTF-8 characters with the MIME type of the item. If the item is content encoded (see below), then the content type refers to the item after content decoding.

`content_encoding` is an optional null-terminated string in UTF-8 characters used to indicate that the binary file is encoded and needs to be decoded before interpreted. The values are as defined for Content-Encoding for HTTP/1.1. Some possible values are “gzip”, “compress” and “deflate”. An empty string indicates no content encoding. Note that the item is stored after the content encoding has been applied.

`extension_type` is a is a printable four-character code that identifies the extension fields of version 1 w.r.t. version 0 of the Item information entry.

`content_location` is a null-terminated string in UTF-8 characters containing the URI of the file as defined in HTTP/1.1 (RFC 2616).

`content_MD5` is a null-terminated string in UTF-8 characters containing an MD5 digest of the file. See HTTP/1.1 (RFC 2616) and RFC 1864.

`content_length` gives the total length (in bytes) of the (un-encoded) file.

`transfer_length` gives the total length (in bytes) of the (encoded) file. Note that transfer length is equal to content length if no content encoding is applied (see above).

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

`group_ID` indicates a file group to which the file item (source file) belongs. See 3GPP TS 26.346 for more details on file groups.

Add the following subclauses before 8.45:

8.44.9 Additional Metadata Container Box

8.44.9.1 Definition

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

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

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

8.44.9.2 Syntax

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

8.44.10 Metabox Relation Box

8.44.10.1 Definition

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

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

8.44.10.2 Syntax

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

8.44.10.3 Sematics

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

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

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

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

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

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

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

Add the following subclauses before Clause 9:

8.46 File Delivery Format Extensions

8.46.1 Introduction

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

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

See Clause 11 for more details on the usage of the file delivery format.

8.46.2 FD Item Information Box

8.46.2.1 Definition

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

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

8.46.2.2 Syntax

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

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

8.46.2.3 Semantics

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

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

8.46.3 File Partition Box

8.46.3.1 Definition

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

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

8.46.3.2 Syntax

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

8.46.3.3 Semantics

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

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

`FEC_encoding_ID` identifies the FEC encoding scheme and is subject to IANA registration (see RFC 3452). Note that i) value zero corresponds to the "Compact No-Code FEC scheme" also known as "Null-FEC" (RFC 3695); ii) value one corresponds to the "MBMS FEC" (3GPP TS 26.346); iii) for values in the range of 0 to 127, inclusive, the FEC scheme is Fully-Specified, whereas for values in the range of 128 to 255, inclusive, the FEC scheme is Under-Specified.

`FEC_instance_ID` provides a more specific identification of the FEC encoder being used for an Under-Specified FEC scheme. This value should be set to zero for Fully-Specified FEC schemes and shall be ignored when parsing a file with `FEC_encoding_ID` in the range of 0 to 127, inclusive. `FEC_instance_ID` is scoped by the `FEC_encoding_ID`. See RFC 3452 for further details.

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

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

`max_number_of_encoding_symbols` gives the maximum number of encoding symbols that can be generated for a source block for those FEC schemes in which the maximum number of encoding symbols is relevant, such as FEC encoding ID 129 defined in RFC 3452. For those FEC schemes in which the maximum number of encoding symbols is not relevant, the semantics of this field is unspecified.

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

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

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

`block_size` indicates the size of a block (in bytes). A block_size that is not a multiple of the encoding_symbol_length symbol size indicates that the last source symbol includes padding that is not stored in the item.

8.46.4 FEC Reservoir Box

8.46.4.1 Definition

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

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

8.46.4.2 Syntax

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

8.46.4.3 Semantics

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

8.46.5 FD Session Group Box

8.46.5.1 Definition

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

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

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

8.46.5.2 Syntax

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

```
        }
        unsigned int(16) num_channels_in_session_group;
        for(k=0; k < num_channels_in_session_group; k++) {
            unsigned int(32) hint_track_id;
        }
    }
}
```

8.46.5.3 Semantics

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

8.46.6 Group ID to Name Box

8.46.6.1 Definition

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

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

8.46.6.2 Syntax

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

8.46.6.3 Semantics

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

8.47 Rate Share Extensions

8.47.1 Introduction

Rate share instructions are used by players and streaming servers to help allocating bitrates dynamically when several streams share a common bandwidth resource. The instructions are stored in the file as sample group entries and apply when scalable or alternative media streams at different bitrates are combined with

other scalable or alternative tracks. The instructions are time-dependent as samples in a track may be associated with different sample group entries. In the simplest case, only one target rate share value is specified per media and time range as illustrated in Figure 4.

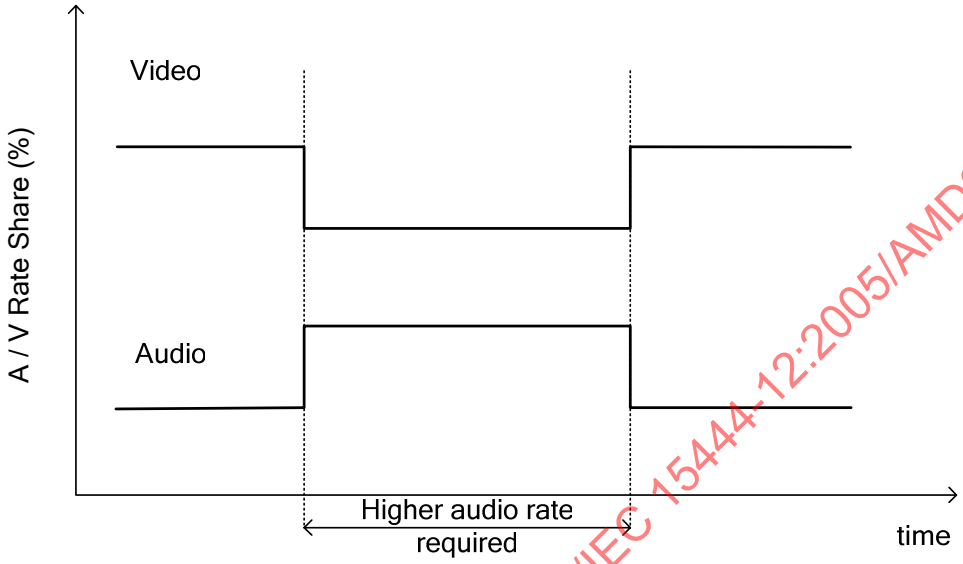

Figure 4 — Audio/Video rate share as function of time

In order to accommodate for rate share values that vary with the available bitrate, it is possible to specify more than one operation range. One may for instance indicate that audio requires a higher percentage (than video) at low available bitrates. Technically this is done by specifying two operation points as shown in Figure 5.

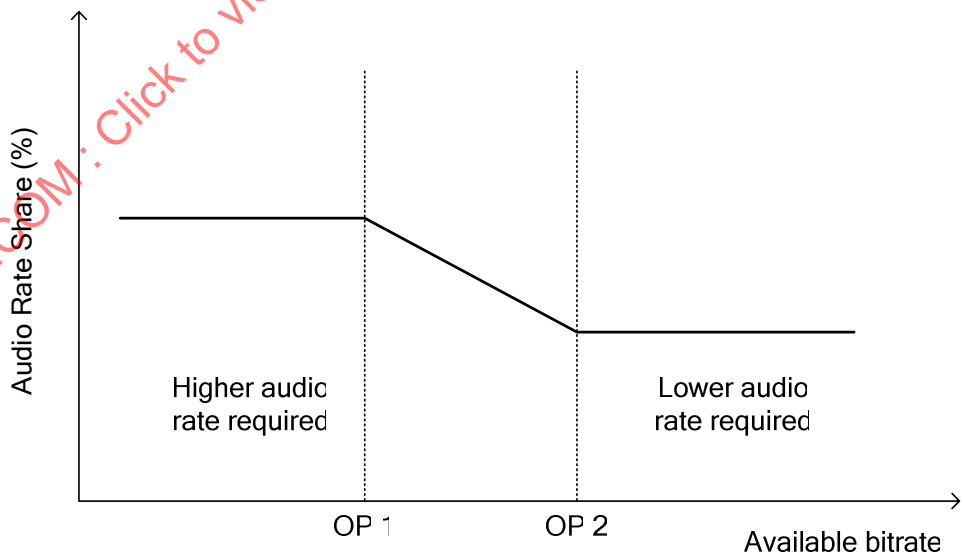

Figure 5 — Audio rate share as function of available bitrate

Operation points are defined in terms of total available bandwidth. For more complex situations it is possible to specify more operation points.

In addition to target rate share values, it is also possible to specify maximum and minimum bitrates for a certain media, as well as discard priority.

8.47.2 Rate Share Sample Group Entry

8.47.2.1 Definition

Each sample of a track may be associated to (zero or) one of a number of sample group descriptions, each of which defines a record of rate-share information. Typically the same rate-share information applies to many consecutive samples and it may therefore be enough to define two or three sample group descriptions that can be used at different time intervals.

The grouping type 'rash' (short for rate share) is defined as the grouping criterion for rate share information. Zero or one sample-to-group box ('sbgp') for the grouping type 'rash' can be contained in the sample table box ('stbl') of a track. It shall reside in a hint track, if a hint track is used, otherwise in a media track.

Target rate share may be specified for several operation points that are defined in terms of the total available bitrate, i.e., the bitrate that should be shared. If only one operation point is defined, the target rate share applies to all available bitrates. If several operation points are defined, then each operation point specifies a target rate share. Target rate share values specified for the first and the last operation points also specify the target rate share values at lower and higher available bitrates, respectively. The target rate share between two operation points is specified to be in the range between the target rate shares of those operation points. One possibility is to estimate with linear interpolation.

8.47.2.2 Syntax

```
class RateShareEntry() extends SampleGroupDescriptionEntry('rash') {
   unsigned int(16)  operation_point_count;
   if (operation_point_count == 1) {
      unsigned int(16)     target_rate_share;
   }
   else {
      for (i=0; i < operation_point_count; i++) {
         unsigned int(32)  available_bitrate;
         unsigned int(16)  target_rate_share;
      }
   }
   unsigned int(32)  maximum_bitrate;
   unsigned int(32)  minimum_bitrate;
   unsigned int(8)   discard_priority;
}
```

8.47.2.3 Semantics

`operation_point_count` is a non-zero integer that gives the number of operation points.

`available_bitrate` is a positive integer that defines an operation point (in kilobits per second). It is the total available bitrate that can be allocated in shares to tracks. Each entry shall be greater than the previous entry.

`target_rate_share` is an integer. A non-zero value indicates the percentage of available bandwidth that should be allocated to the media for each operation point. The value of the first (last) operation point applies to lower (higher) available bitrates than the operation point itself. The target rate share between operation points is bounded by the target rate shares of the corresponding operation points. A zero value indicates that no information on the preferred rate share percentage is provided.

`maximum_bitrate` is an integer. A nonzero value indicates (in kilobits per second) an upper threshold for which bandwidth should be allocated to the media. A higher bitrate than maximum bitrate should only be allocated if all other media in the session has fulfilled their quotas for target rate-share and

maximum bitrate, respectively. A zero value indicates that no information on maximum bitrate is provided.

`minimum_bitrate` is an integer. A nonzero value indicates (in kilobits per second) a lower threshold for which bandwidth should be allocated to the media. If the allocated bandwidth would correspond to a smaller value, then no bitrate should be allocated. Instead preference should be given to other media in the session or alternate encodings of the same media. Zero minimum bitrate indicates that no information on minimum bitrate is provided.

`discard_priority` is an integer indicating the priority of the track when tracks are discarded to meet the constraints set by target rate share, maximum bitrate and minimum bitrate. Tracks are discarded in discard priority order and the track that has the highest discard priority value is discarded first.

8.47.3 Relationship between tracks

The purpose of defining rate share information is to aid a server or player extracting data from a track in combination with other tracks. Note that a server/player streams/plays tracks simultaneously if they belong to different alternate groups and can switch between tracks that belong to the same switch group within an alternate group. By default, all tracks are served/played simultaneously if no alternate groups are defined.

Rate share information should be provided for each track. A track that does not include rate share information has one operation point and can be treated as a constant-bitrate track with discard priority 128. Target rate share, minimum and maximum bitrates do not apply in this case.

Tracks that are alternates to each other shall (at each instance of time) define the same number of operation points at the same set of total available bitrates and have the same discard priorities. Note that the number and definition of operation points may depend on time. Alternate tracks may have different target rate shares, minimum and maximum bitrates.

8.47.4 Bitrate allocation

Rate share information on maximum bitrate, minimum bitrate, and target rate share can be combined for a track. If this is the case, the target rate share shall be applied to find an allocated bitrate before the impact of the maximum and minimum bitrates is considered.

When allocating bandwidth to several tracks, the following considerations apply:

1. In the case all tracks have explicit target rate share values and they don't sum up to 100 per cent, treat them as weights, i.e., normalize them.
2. The total allocation shall not exceed total available bitrate.
3. In a choice between alternate tracks, the chosen track should be the track that causes the alternate group to have an allocation most closely in accord with its target rate share, or the track that desires the highest bitrate that can be allocated without discarding other tracks (see below).
4. Tracks must have an allocation between their minimum and maximum bitrates, or be discarded.
5. Tracks should have an allocation in accord with their target rate shares, but this may be distorted to allow some tracks to achieve their minima, or in case some have reached their maxima.
6. If an allocation cannot be done including a track from every alternate group, then tracks should be discarded in discard priority order.
7. The allocation must be re-calculated whenever the operating set for an active track (one that has been selected from an alternate group) changes or the available bitrate changes.

8.48 Grouping and labelling of tracks

8.48.1 Introduction

A typical presentation stored in a file contains one alternate group per media type: one for video, one for audio, etc. Such a file may include several video tracks, although, at any point in time, only one of them

should be played or streamed. This is achieved by assigning all video tracks to the same alternate group. (See subclause 8.5 for the definition of alternate groups.)

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

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

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

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

8.48.2 Track Selection Box

8.48.2.1 Definition

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

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

8.48.2.2 Syntax

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

8.48.2.3 Semantics

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

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

8.48.2.4 Attributes

The following attributes are descriptive:

Name	*Attribute*	*Description*
Temporal scalability	'tesc'	The track can be temporally scaled.
Fine-grain SNR scalability	'fgsc'	The track can be fine-grain scaled.
Coarse-grain SNR scalability	'cgsc'	The track can be coarse-grain scaled.
Spatial scalability	'spsc'	The track can be spatially scaled.
Region-of-interest scalability	'resc'	The track can be region-of-interest scaled.

The following attributes are differentiating:

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

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

Add the following clause after 10.5:

11 ALC/LCT and FLUTE Hint Track Format

11.1 Introduction

The file format supports multicast/broadcast delivery of files with FEC protection. Files to be delivered are stored as items in a container file (defined by the file format) and the meta box containing these so-called source files is amended with information on how the files are partitioned into source symbols. For each source block of a FEC encoding, additional parity symbols can pre-computed and stored as FEC reservoir items. The partitioning depends on the FEC scheme, the target packet size, and the desired FEC overhead. The actual transmission is governed by hint tracks that contain server instructions that facilitate the encapsulation of source and FEC symbols into packets. The main architectural difference between File Delivery (FD) hint tracks and streaming hint tracks is that the latter can refer to meta box items in addition to samples in tracks.

FD hint tracks have been designed for the ALC/LCT (Asynchronous Layered Coding/Layered Coding Transport) and FLUTE (File Delivery over Unidirectional Transport) protocols. LCT provides transport level support for reliable content delivery and stream delivery protocols. ALC is a protocol instantiation of the LCT

building block, and it serves as a base protocol for massively scalable reliable multicast distribution of arbitrary binary objects. FLUTE builds on top of ALC/LCT and defines a protocol for unidirectional delivery of files.

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

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

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

The syntax for using the meta box as a container file for source files is defined in subclause 8.44, partitions and FEC reservoirs are defined in subclause 8.46, while the syntax for FD hint tracks is defined in subclauses 11.3 and 11.4.

11.2 Design principles

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

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

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

A schematic picture of a file containing three alternative hint tracks with different FEC overhead for a source file is provided in Figure 6.

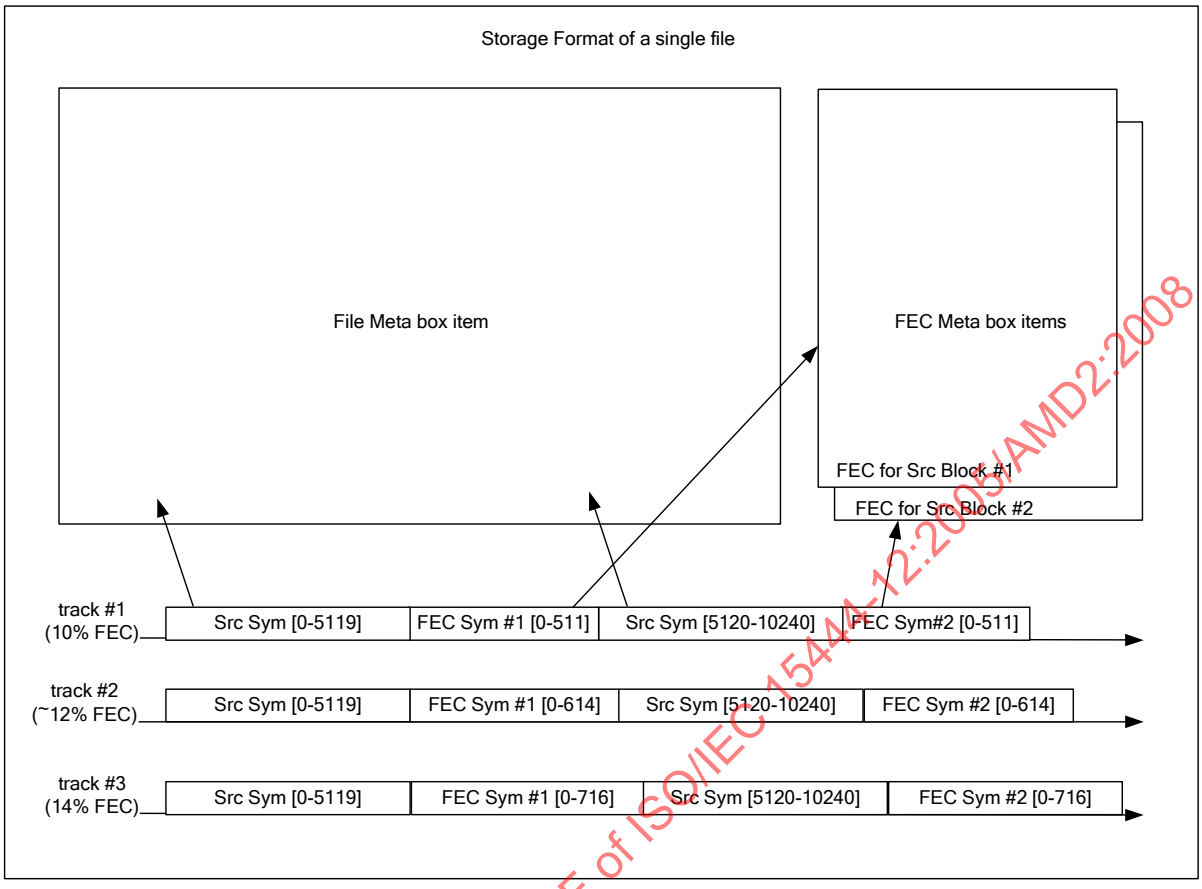

Figure 6 —Different FEC overheads of a source file provided by alternative hint tracks.

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

11.3 Sample Description Format

11.3.1 Definition

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

11.3.2 Syntax

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

11.3.3 Semantics

`partition_entry_ID` indicates the partition entry in the FD item information box. A zero value indicates that no partition entry is associated with this sample entry, e.g., for FDT.

`FEC_overhead` is a fixed 8.8 value indicating the percentage protection overhead used by the hint sample(s). The intention of providing this value is to provide characteristics to help a server select a session group (and corresponding FD hint tracks).

The `hinttrackversion` and `highestcompatibleversion` fields have the same interpretation as in the RTP hint sample entry described in subclause 10.2. As additional data a time scale entry box may be provided. If not provided, there is no indication given on timing of packets.

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

11.4 Sample Format

11.4.1 Sample Container

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

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

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

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

11.4.2 Packet Entry Format

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

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

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

11.4.3 LCT Header Template Format

The LCT header template is defined as follows:

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

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

11.4.4 LCT Header Extension Constructor Format

The LCT header extension constructor format is defined as follows:

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

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

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

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

Note that a server can identify packets including FDT by observing whether EXT_FDT (header_extension_type == 192) is present.

11.4.5 Packet Constructor Format

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

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

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

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

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

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

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

11.4.6 Extra Data Box

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

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

In Annex B, replace:

Ericsson

with:

Telefonaktiebolaget LM Ericsson