INTERNATIONAL STANDARD ISO/IEC 14496-3:1999/Amd.1:2000
TECHNICAL CORRIGENDUM 1

Published 2001-08-01

INTERNATIONAL ORGANIZATION FOR STANDARDIZATION • МЕЖДУНАРОДНАЯ ОРГАНИЗАЦИЯ ПО СТАНДАРТИЗАЦИИ • ORGANISATION INTERNATIONALE DE NORMALISATION
INTERNATIONAL ELECTROTECHNICAL COMMISSION • МЕЖДУНАРОДНАЯ ЭЛЕКТРОТЕХНИЧЕСКАЯ КОМИССИЯ • COMMISSION ÉLECTROTECHNIQUE INTERNATIONALE

Information technology — Coding of audio-visual objects —

Part 3:
Audio

AMENDMENT 1: Audio extensions

TECHNICAL CORRIGENDUM 1

Technologies de l'information — Codage des objets audiovisuels —

Partie 3: Codage audio

AMENDEMENT 1: Extensions audio

RECTIFICATIF TECHNIQUE 1

Technical Corrigendum 1 to International Standard ISO/IEC 14496-3:1999/Amd.1:2000 was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee SC 29, *Coding of audio, picture, multimedia and hypermedia information*.

ICS 35.040 **Ref. No. ISO/IEC 14496-3:1999/Amd.1:2000/Cor.1:2001(E)**

Printed in Switzerland

Throughout the text of ISO/IEC 14496-3:1999/Amd.1:2000, replace all occurrences of "*AL PDU*" with "*SL packet*" and all occurrences of "*alPduPayload*" with "*slPacketPayload*".

In clause "Introduction", add

"

"MPEG-4 has no standard for transport. In all of the MPEG-4 tools for audio and visual coding, the coding standard ends at the point of constructing a sequence of access units that contain the compressed data. The MPEG-4 Systems (ISO/IEC 14496-1:2001) specification describes how to convert the individually coded objects into a bitstream that contains a number of multiplexed sub-streams.

There is no standard mechanism for transport of this stream over a channel; this is because the broad range of applications that can make use of MPEG-4 technology have delivery requirements that are too wide to easily characterize with a single solution. Rather, what is standardized is an interface (the Delivery Multimedia Interface Format, or DMIF, specified in ISO/IEC 14496-6:1999) that describes the capabilities of a transport layer and the communication between transport, multiplex, and demultiplex functions in encoders and decoders. The use of DMIF and the MPEG-4 Systems bitstream specification allows transmission functions that are much more sophisticated than are possible with previous MPEG standards.

However, LATM and LOAS have been defined to provide a Low overhead Audio multiplex and transport mechanism for natural audio applications, which do not require sophisticated object-based coding or other functions provided by MPEG-4 Systems.

The following table gives an overview about the multiplex, storage and transmission formats for MPEG-4 Audio currently available within the MPEG-4 framework:

	Format	Functionality defined in:	Functionality redefined in:	Description
Multiplex	FlexMux	ISO/IEC 14496-1:2001 (MPEG-4 Systems) (Normative)		Flexible multiplex scheme
	LATM	ISO/IEC 14496-3/Amd.1:2000 (MPEG-4 Audio V2) (Normative)		Low Overhead Audio Transport Multiplex
Storage	ADIF	ISO/IEC 13818-7:1997 (MPEG-2 Audio) (Normative)	ISO/IEC 14496-3:1999 (MPEG-4 Audio V1) (Informative)	(MPEG-2 AAC) Audio Data Interchange Format, AAC only
	MP4FF	ISO/IEC 14496-1/Amd. 1:2000 (MPEG-4 Systems V2) (Normative)		MPEG-4 File format
Transmission	ADTS	ISO/IEC 13818-7:1997 (MPEG-2 Audio) (Normative, Exemplarily)	ISO/IEC 14496-3:1999 (MPEG-4 Audio V1) (Informative)	Audio Data Transport Stream, AAC only
	LOAS	ISO/IEC I14496-3/Amd. 1:2000 (MPEG-4 Audio V2) (Normative, Exemplarily)		Low Overhead Audio Stream, based on LATM, three versions are available: AudioSyncStream() EPAudioSyncStream() AudioPointerStream()

".

Note: This text is intended to replace the first bullet in subclause 1.1.2 (New concepts in MPEG-4 Audio) as stated in MPEG-4 Audio (ISO/IEC 14496-3:1999).

Replace the first row of Table 3 – Complexity of Audio Object Types in subclause 5.2.2 with

"

Object Type	Parameters	PCU (MOPS per channel)	RCU (kWords per channel)	Remarks

".

*In subclause 5.2.3, replace within table footnotes *1 below Table 4 (Levels for the High Quality Audio Profile), Table 5 (Levels for the Low Delay Audio Profile), Table 6 (Levels for the Natural Audio Profile) and Table 7 (Levels for the Mobile Audio Internetworking Profile)*

"

..., which has the longest frame length, in each profile & level.

"

with

"

..., which has the longest maximum frame length, in each profile & level. For audio object types supporting variable frame lengths and arbitrary bitrates (i.e. any AAC audio object type) this does just extend the required decoder input buffer but does not affect the amount of redundancy actually applied.

".

In Table 8 (Syntax of AudioSpecificInfo()) in subclause 6.2.1, replace

"

epConfig; if (epConfig == 2) ErrorProtectionSpecificConfig();	**2**	**bslbf**

"

with

"

epConfig;
if (epConfig == 2 || epConfig == 3) {
ErrorProtectionSpecificConfig();
}
if (epConfig == 3) {
directMapping;
if (! directMapping) {
/* tbd */
}
}
	2
	
	1
		uimsbf
		
		bslbf

".

In subclause 6.3.5, replace

"

This variable signals what kind of error robust configuration is used, i. e. how instances of error sensitivity categories are obtained on decoder site.

Table 10: epConfig

epConfig	Description
0	All instances of all sensitivity categories belonging to one frame are stored within one access unit.
1	Each instance of each sensitivity category belonging to one frame is stored separately within a single access unit, i.e. there are as many elementary streams existent as instances defined within a frame.
2	The error protection decoder has to be applied. Its input is an error protected access unit and its output are several error protection class instances. Each instance of each sensitivity category belonging to one frame corresponds to one of these error protection class instances.
3	Reserved

"

with

"

This data element signals what kind of error robust configuration is used.

Table 10: epConfig

epConfig	Description
0	All instances of all error sensitivity categories belonging to one frame are stored within one access unit. There exists one elementary stream per scalability layer, or just one elementary stream in case of non-scalable configurations.
1	Each instance of each sensitivity category belonging to one frame is stored separately within a single access unit, i.e. there exist as many elementary streams as instances defined within a frame.
2	The error protection decoder has to be applied. Its input are error protected access units. There exists one elementary stream per scalability layer, or just one elementary stream in case of non-scalable configurations. The output of the EP decoder is a set of several error protection classes. The concatenation of EP classes at the error protection decoder output is equivalent to epConfig=0 data.
3	The error protection decoder has to be applied. Its input are error protected access units. There exists one elementary stream per scalability layer, or just one elementary stream in case of non-scalable configurations. The output of the EP decoder is a set of several error protection classes. The concatenation of EP classes at the error protection decoder output is equivalent to epConfig=0 data. The mapping between EP classes and ESC instances is signaled by the data element directMapping.

".

After subclause 6.3.5, add

"

6.3.6 direct mapping

This data element identifies the mapping between error protection classes and error sensitivity category instances.

directMapping

directMapping	Description
0	Reserved
1	Each error protection class is treated as an instance of an error sensitivity category (one to one mapping), so that the error protection decoder output is equivalent to epConfig=1 data.

".

To the end of subclause 6.5.1, add

"

The mechanism defined in this subclause should not be used for transmission of TTSI object (12), Main Synthetic object (13), Wavetable Synthesis object (14), General MIDI object (15) and Algorithmic Synthesis and Audio FX object (16). For these object types, other multiplex and transport mechanisms might be used, e.g. those defined in MPEG-4 Systems.

".

To the description of audioMuxElementStartPointer in subclause 6.5.2.2, add

"

The maximum possible value of this field is reserved to signal that there is no start of an access unit in this sync frame.

".

Replace Table 17 (Syntax of AudioMuxElement()) in subclause 6.5.3.1 (Syntax) with

"

Syntax	No. of bits	Mnemonic
AudioMuxElement(muxConfigPresent)		
{		
if(muxConfigPresent) {		
useSameStreamMux;	**1**	**bslbf**
if (!useSameStreamMux)		
StreamMuxConfig();		
}		
if (audioMuxVersion == 0) {		
for(i=0; i<=numSubFrames; i++) {		
PayloadLengthInfo();		
PayloadMux();		
}		
if(otherDataPresent) {		
for(i=0; i<otherDataLenBits; I++) {		
otherDataBit;	**1**	**bslbf**
}		
}		
}		
else {		
/* tbd */		
}		
}		

".

Replace table 18 (Syntax of StreamMuxConfig) in subclause 6.5.3.1 (Syntax) with

"

Syntax	No. of bits	Mnemonic
StreamMuxConfig()		
{		
audioMuxVersion;	**1**	**bslbf**
if (audioMuxVersion == 0) {		
streamCnt = 0;		
allStreamsSameTimeFraming;	**1**	**uimsbf**
numSubFrames;	**6**	**uimsbf**
numProgram;	**4**	**uimsbf**
for (prog = 0; prog <= numProgram; prog++) {		
numLayer;	**3**	**uimsbf**
for (lay = 0; lay <= numLayer; lay++) {		
progSIndx[streamCnt]=prog; laySIndx[streamCnt]=lay;		
streamID [prog][lay] = streamCnt++;		
if (prog == 0 & lay == 0) {		
AudioSpecificInfo();		
}else {		
useSameConfig;	**1**	**uimsbf**
if (!useSameConfig)		
AudioSpecificInfo();		
}		
frameLengthType[streamID[prog][lay]];	**3**	**uimsbf**
if (frameLengthType[streamID[prog][lay] == 0) {		
bufferFullness[streamID[prog][lay]];	**8**	**uimsbf**
if (! allStreamsSameTimeFraming) {		
if ((AudioObjectType[lay]==6 ||		
AudioObjectType[lay]== 20) &&		
(AudioObjectType[lay-1]==8 ||		
AudioObjectType[lay-1]==24)) {		
coreFrameOffset;	**6**	**uimsbf**
}		
}		
} else if(frameLengthType[streamID[prog][lay]] == 1) {		
frameLength[streamID[prog][lay]];	**9**	**uimsbf**
} else if (frameLengthType[streamID[prog][lay]] == 4		
frameLengthType[streamID[prog][lay]] == 5		
frameLengthType[streamID[prog][lay]] == 3) {		
CELPframeLengthTableIndex[streamID[prog][lay]];	**6**	**uimsbf**
} else if (frameLengthType[streamID[prog][lay]] == 6		
frameLengthType[streamID[prog][lay]] == 7) {		
HVXCframeLengthTableIndex[streamID[prog][lay]];	**1**	**uimsbf**
}		
}		
}		
otherDataPresent;	**1**	**uimsbf**
if (otherDataPresent) {		
otherDataLenBits = 0; /* helper variable 32bit */		
do {		
otherDataLenBits = otherDataLenBits * 2^8;		
otherDataLenEsc;	**1**	**uimsbf**
otherDataLenTmp;	**8**	**uimsbf**
otherDataLenBits = otherDataLenBits + otherDataLenTmp;		
} while (otherDataLenEsc);		

```
        }
        crcCheckPresent;                                    1       uimsbf
        if( crcCheckPresent ) crcCheckSum;                  8       uimsbf
    }
    else {
        /* tbd */
    }
}
```

".

Replace Table 19 (Syntax of PayloadLengthInfo()) in subclause 6.5.3.1 (Syntax) with

"

Syntax	No. of bits	Mnemonic
PayloadLengthInfo()		
{		
if(allStreamsSameTimeFraming) {		
for (prog = 0; prog <= numProgram; prog++) {		
for (lay = 0; lay <= numLayer; lay++) {		
if(frameLengthType[streamID[prog][lay]] == 0) {		
do { /* always one complete access unit */		
tmp;	**8**	**uimsbf**
MuxSlotLengthBytes[streamID[prog][lay]] += tmp;		
} while(tmp==255);		
} else {		
if (frameLengthType[streamID[prog][lay]] == 5 ||		
frameLengthType[streamID[prog][lay]] == 7 ||		
frameLengthType[streamID[prog][lay]] == 3) {		
MuxSlotLengthCoded[streamID[prog][lay]];	**2**	**uimsbf**
}		
}		
}		
}		
} else {		
numChunk;	**4**	**uimsbf**
for (chunkCnt=0; chunkCnt <= numChunk; chunkCnt++) {		
streamIndx;	**4**	**uimsbf**
prog = progCIndx[chunkCnt] = progSIndx[streamIndx]**;**		
lay = layCIndx[chunkCnt] = laySIndx [streamIndx];		
if(frameLengthType[streamID[prog][lay]] == 0) {		
do { /* not necessarily a complete access unit */		
tmp;	**8**	**uimsbf**
MuxSlotLengthBytes[streamID[prog][lay]] += tmp;		
} while (tmp == 255);		
AuEndFlag[streamID[prog][lay]];	**1**	**bslbf**
} else {		
if (frameLengthType[streamID[prog][lay]] == 5 ||		
frameLengthType[streamID[prog][lay]] == 7 ||		
frameLengthType[streamID[prog][lay]] == 3) {		
MuxSlotLengthCoded[streamID[prog][lay]];	**2**	**uimsbf**
}		
}		
}		
}		
}		

".

Replace Table 20 (Syntax of PayloadMux()) in subclause 6.5.3.1 (Syntax) with

"

Syntax	No. of bits	Mnemonic

```
PayloadMux()
{
    if( allStreamsSameTimeFraming ) {
        for ( prog = 0; prog <= numProgram; prog++ ) {
            for ( lay = 0; lay <= numLayer; lay++ ) {
                payload [streamID[prog][ lay]];
            }
        }
    } else {
        for (chunkCnt=0; chunkCnt <= numChunk; chunkCnt++) {
            prog = progCIndx[chunkCnt];
            lay = layCIndx  [chunkCnt];

            payload [streamID[prog][ lay]];
        }
    }
}
```

".

In subclause 6.5.3.2 (Semantics), add

"

audioMuxVersion: A data element to signal the bitstream syntax version. possible values: 0 (default), 1 (reserved for future extensions).

".

In subclause 6.5.3.2 (Semantics), replace

"

Note 1: There might be more than one instance of error sensitivity category 1 and 2 depending on the value of the variable **numSubFrames** defined in **StreamMuxConfig()**. Figure 3 shows an example for the order of the instances assuming numSubFrames is two (2).

"

with

"

Note 1: There might be more than one instance of error sensitivity category 1 and 2 depending on the value of the variable **numSubFrames** defined in **StreamMuxConfig()**. Figure 3 shows an example for the order of the instances assuming numSubFrames is one (1).

".

In subclause 6.5.3.2 (Semantics), replace

"

numSubFrames A field indicating how many PayloadMux() frames are multiplexed. If more than one PayloadMux() frame are multiplexed, all PayloadMux() share a common StreamMuxConfig().The minimum value is 1.

numProgram A field indicating how many programs are multiplexed. The minimum value is 1.

numLayer A field indicating how many scalable layers are multiplexed. The minimum value is 1.

"

with

"

numSubFrames A field indicating how many PayloadMux() frames are multiplexed (numSubFrames+1). If more than one PayloadMux() frame are multiplexed, all PayloadMux() share a common StreamMuxConfig().The minimum value is 0 indicating 1 subframe.

numProgram A field indicating how many programs are multiplexed (numProgram+1). The minimum value is 0 indicating 1 program.

numLayer A field indicating how many scalable layers are multiplexed (numLayer+1). The minimum value is 0 indicating 1 layer.

".

In subclause 6.5.3.2 (Semantics), replace

"

numChunk A field indicating the number of payload chunks. Each chunk may belong to an access unit with a different time base; only used if allStreamsSameTimeFraming is set to zero. The minimum value is 1.

"

with

"

numChunk A field indicating the number of payload chunks (numChunk+1). Each chunk may belong to an access unit with a different time base; only used if allStreamsSameTimeFraming is set to zero. The minimum value is 0 indicating 1 chunk.

".

In subclause 6.5.3.2 (Semantics), replace

"

otherDataLenBits A field indicating the length of the other data.

"

with

"

otherDataLenBits A helper variable indicating the length in bits of the other data.

otherDataLenEsc A field indicating whether there is more than one otherDataLenTmp data element following.

otherDataLenTmp A field used to calculate otherDataLenBits.

".

In subclause 6.5.3.2, replace

"

blockDelay A field indicating the time base of the payload with frameLengthType of 0. The time base is specified by a two-layer scheme. blockDelay provides a coarse time base in multiples of tb1 = frame_length_samples / sampling_rate. If enhanced time resolution is desired, fractionalDelay may specify an additional fractional value.

fractionalDelayPresent A flag indicating the presence of fractionalDelay in the current payload.

fractionalDelayPresent	Description
0	fractionalDelay is not present.
1	fractionalDelay is present.

fractionalDelay A field indicating the enhanced time resolution for the time base of the payload with frameLengthType of 0.

"

with

"

bufferFullness state of the bit reservoir. It is transmitted as the number of available bits in the bit reservoir divided by the number of audio channels divided by 32 and truncated to an integer value (mean number of 32 bit words per channel remaining in the encoder buffer after encoding the first audio frame in the AudioMuxElement()). A value of hexadecimal FF signals that the bitstream is a variable rate bitstream. In this case, buffer fullness is not applicable.

coreFrameOffset identifies the first CELP frame of the current super-frame. It is defined only in case of scalable configurations with CELP core and AAC enhancement layer(s) and transmitted with the first AAC enhancement layer. The value 0 identifies the first CELP frame following StreamMuxConfig() as the first CELP frame of the current super-frame. A value > 0 signals the number of CELP frames that the first CELP frame of the current super-frame is transmitted earlier in the bitstream.

Usage of LATM in case of scalable configurations with CELP core and AAC enhancement layer(s):

— Instances of the AudioMuxElement() are transmitted in equidistant manner.

— The represented timeframe of one AudioMuxElement() is similar to a multiple of a super-frame timeframe.

— The relative number of bits for a certain layer within any AudioMuxElement() compared to the total number of bits within this AudioMuxElement() is equal to the relative bitrate of that layer compared to the bitrate of all layers.

— In case of coreFrameOffset=0 and bufferFullness=0, all core coder frames and all AAC frames of a certain super-frame are stored within the same instance of AudioMuxElement().

— In case of coreFrameOffset>0, several or all core coder frames are stored within previous instances of AudioMuxElement().

— Any core layer related configuration information refers to the core frames transmitted within the current instance of the AudioMuxElement(), independent of the value of coreFrameOffset.

— A specified bufferFullness is related to the first AAC frame of the first super-frame stored within the current AudioMuxElement().

— The value of bufferFullness can be used to determine the location of the first bit of the first AAC frame of the current layer of the first super-frame stored within the current AudioMuxElement() by means of a backpointer: $backPointer = -meanFrameLength + bufferFullness + currentFrameLength$

The backpointer value specifies the location as a negative offset from the current AudioMuxElement(), i. e. it points backwards to the beginning of an AAC frame located in already received data. Any data not belonging to the payload of the current AAC layer is not taken into account. If bufferFullness=='0', then the AAC frame starts after the current AudioMuxElement().

".

In subclause 7.3.1, replace

"

Parametric Base Layer -- Configuration

For the parametric coder in unscalable mode or for the base layer in HILN scalable mode the following ParametricSpecificConfig() is required:

```
ParametricSpecificConfig() {
        PARAconfig();
}
```


Parametric HILN Enhancement / Extension Layer -- Configuration

To use HILN as core in an "T/F scalable with core" mode, in addition to the HILN base layer an HILN enhancement layer is required. In HILN bitrate scalable operation, in addition to the HILN base layer one or more HILN extension layers are permitted. Both the enhancement layer and the extension layer have the following ParametricSpecificConfig():

```
ParametricSpecificConfig() {
        HILNenexConfig();
}
```

"

with

"

Parametric Base Layer -- Configuration

The parametric coder in unscalable mode or the base layer in HILN scalable mode uses a ParametricSpecificConfig() with isBaseLayer==1.

Parametric HILN Enhancement / Extension Layer -- Configuration

To use HILN as core in a "T/F scalable with core" mode, in addition to the HILN base layer an HILN enhancement layer is required. In HILN bitrate scalable operation, in addition to the HILN base layer one or more HILN extension layers are permitted. Both the enhancement and extension layer use a ParametricSpecificConfig() with isBaseLayer==0.

```
ParametricSpecificConfig() {
        isBaseLayer;                                    1 bit   uimsbf
        if ( isBaseLayer ) {
                PARAconfig();
        }
        else {
                HILNenexConfig();
        }
}
```


",

and at the end of subclause 7.3.1, add

"

isBaseLayer A one-bit identifier representing whether the corresponding layer is the base layer (1) or an enhancement or extension layer (0).

".

In subclause 7.3.2.3, replace

"

Table 72: LARH3 code (harmLAR[7..25])

"

with

"

Table 72: LARH3 code (harmLAR[7..24])

",

and replace
"

Table 74: LARN2 code (noiseLAR[2..25])

"
with
"

Table 74: LARN2 code (noiseLAR[2..24])

".

In Table 25 and in 7.4.1.1, replace " extensionFlag *" with "* PARAextensionFlag *".*

In subclause in 7.5.1.4.3, add
"
If the speed is controlled by the time scaling factor in the **speed** field of the AudioSource BIFS node, the speed change factor is:

speedFactor = 1 / speed;

If the pitch is controlled by the **pitch** field of the AudioSource BIFS node, the pitch change factor is:

pitchFactor = pitch;

"

after
"
If playback at the original speed and pitch is desired, the corresponding control factors are set to their default values:

speedFactor = 1.0;

pitchFactor = 1.0;

".