INTERNATIONAL STANDARD ISO/IEC 14496-2:1999
TECHNICAL CORRIGENDUM 1

Published 2000-08-15

Information technology — Coding of audio-visual objects —

Part 2: **Visual**

TECHNICAL CORRIGENDUM 1

Technologies de l'information — Codage des objets audiovisuels —

Partie 2: Codage visuel

RECTIFICATIF TECHNIQUE 1

Technical Corrigendum 1 to International Standard ISO/IEC 14496-2:1999 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 14496-2:1999/Cor.1:2000(E)**

Throughout the whole document, replace "quantization" with "quantisation".

On Page xiii, Overview of the object based nonscalable syntax, replace the following paragraph:

"

The coded representation defined in the non-scalable syntax achieves a high compression ratio while preserving good image quality. Further, when access to individual objects is desired, the shape of objects also needs to be coded, and depending on the bandwidth available, the shape information can be coded lossy or losslessly.

"

with

"

The coded representation defined in the non-scalable syntax achieves a high compression ratio while preserving good image quality. Further, when access to individual objects is desired, the shape of objects also needs to be coded, and depending on the bandwidth available, the shape information can be coded in a lossy or lossless fashion.

"

In Subclause 5.1, Method of describing bitstream syntax, replace the following table:

"

while (condition) {
    data_element
    . . .
}
	If the condition is true, then the group of data elements occurs next in the data stream. This repeats until the condition is not true.

do {
    data_element
    . . .
} while (condition)
	The data element always occurs at least once.
	
	The data element is repeated until the condition is not true.

if (condition) {
    data_element
    . . .
} else {
    data_element
    . . .
}
	If the condition is true, then the first group of data elements occurs next in the data stream.
	
	If the condition is not true, then the second group of data elements occurs next in the data stream.

for (i = m; i < n; i++) {
    data_element
    . . .
}
	The group of data elements occurs (n-m) times. Conditional constructs within the group of data elements may depend on the value of the loop control variable i, which is set to m for the first occurrence, incremented by one for the second occurrence, and so forth.

/* comment **. . .** */	Explanatory comment that may be deleted entirely without in any way altering the syntax.

"

with

"

while (condition) {
    data_element
    . . .
}
	If the condition is true, then the group of data elements occurs next in the data stream. This repeats until the condition is not true.

do {
    data_element
    . . .
} while (condition)
	The data element always occurs at least once.
	
	The data element is repeated until the condition is not true.

do {	
. . .	
continue	The continue continues execution of the next repetition of the nearest while-do loop.
. . .	

} while (condition)	

if (condition) {
    data_element
    . . .
} else {
    data_element
    . . .
}
	If the condition is true, then the first group of data elements occurs next in the data stream.
	
	If the condition is not true, then the second group of data elements occurs next in the data stream.

for (i = m; i < n; i++) {
    data_element
    . . .
}
	The group of data elements occurs (n-m) times. Conditional constructs within the group of data elements may depend on the value of the loop control variable i, which is set to m for the first occurrence, incremented by one for the second occurrence, and so forth.

/* comment **. . .** */	Explanatory comment that may be deleted entirely without in any way altering the syntax.

"

In Subclause 5.2.4, Definition of next_start_code() function, delete the following paragraph:

"

This function checks whether the current position is byte aligned. If it is not, a zero stuffing bit followed by a number of one stuffing bits may be present before the start code.

"

In Clause 3, Definitions, add the following definitions with the appropriate numbering in alphabetical order:

"

3.xxx **mesh object planes**, **MOP:** The instance of mesh objects at a given time.
3.xxx **video object planes**, **VOP:** The instance of video objects at a given time.

"

In Clause 3, Definitions, remove the following definition:

"

3.3 **backward compatibility**: A newer coding standard is backward compatible with an older coding standard if decoders designed to operate with the older coding standard are able to continue to operate by decoding all or part of a bitstream produced according to the newer coding standard.

"

In Table 6-3, replace the following row 5 (left column):

“

visual_object_sequence__start_code

"

with

“

visual_object_sequence_start_code

"

In Subclause 6.2.1, Start Codes, replace the following paragraph:

"

When coded visual objects are carried within a Systems bitstream defined by ISO/IEC 14496-1, configuration information and elementary stream data are always carried separately. Configuration information and elementary streams follow the syntax below, subject to the break points between them defined above. The Systems specification ISO/IEC 14496-1 defines containers that are used to carry Visual Object and Visual Object Layer configuration information. A separate container is used for each object. For video objects, a separate container is also used for each layer. VisualObjectSequence headers are not carried explicitly, but the information is contained in other parts of the Systems bitstream.

"

with

"

When coded visual objects are carried within a Systems bitstream defined by ISO/IEC 14496-1, configuration information and elementary stream data are always carried separately. Configuration information and elementary streams follow the syntax below, subject to the break points between them defined above. The Systems specification ISO/IEC 14496-1 defines containers that are used to carry Visual Object Sequence, Visual Object and Video Object Layer configuration information. For video objects one container is used for each layer for each object. This container carries a Visual Object Sequence header, a Visual Object header and a Video Object Layer header. For other types of visual objects, one container per visual object is used. This container carries a Visual Object Sequence header and a Visual Object header. The Visual Object Sequence Header must be identical for all visual streams input simultaneously to a decoder. The Visual Object Headers for each layer of a multilayer object must be identical.

"

In Subclause 6.2.1, Start Codes, replace the following paragraph:

"

The elementary stream data associated with a single layer may be wrapped in configuration information defined in accordance with the syntax below. A visual bitstream may contain at most one instance of each of VisualObjectSequence(), VisualObject() and VideoObjectLayer(). The Visual Object Sequence Header must be identical for all streams input simultaneously to a decoder. The Visual Object Headers for each layer of a multilayer object must be identical.

"

with

"

The elementary stream data associated with a single layer may be wrapped in configuration information defined in accordance with the syntax below. A visual bitstream may contain at most one instance of each of VisualObjectSequence(), VisualObject() and VideoObjectLayer(), with the exception of repetition of the Visual Object Sequence Header, the Visual Object Header and the Video Object Layer Header as described below. The Visual Object Sequence Header must be identical for all visual streams input simultaneously to a decoder. The Visual Object Headers for each layer of a multilayer object must be identical. The Visual Object Sequence Header, the Visual Object Header and the Video Object Layer Header may be repeated in a single visual bitstream. Repeating these headers enables random access into the visual bitstream and recovery of these headers when the original headers are corrupted by errors. This header repetition is used only when visual_object_type in the Visual Object Header indicates that visual object type is video. (i.e. visual_object_type=="video ID") All of the data elements in the Visual Object Sequence Header, the Visual Object Header and the Video Object Layer Header repeated in a visual bitstream shall have the same value as in the original headers, except that first_half_vbv_occupancy and latter_half_vbv_occupancy may be changed to specify the VBV occupancy just before the removal of the first VOP following the repeated Video Object Layer Header.

"

In Subclause 6.2.2, Visual Object Sequence and Visual Object, replace the VisualObjectSequence() syntax:

"

VisualObjectSequence() {	**No. of bits**	**Mnemonic**
visual_object_sequence_start_code	32	bslbf
profile_and_level_indication	8	uimsbf
while (next_bits()== user_data_start_code){		
user_data()		
}		
VisualObject()		
visual_object_sequence_end_code	32	bslbf
}		

"

with
"

VisualObjectSequence() {	**No. of bits**	**Mnemonic**
do {		
visual_object_sequence_start_code	32	bslbf
profile_and_level_indication	8	uimsbf
while (next_bits()== user_data_start_code){		
user_data()		
}		
VisualObject()		
} while (next_bits() != visual_object_sequence_end_code)		
visual_object_sequence_end_code	32	bslbf
}		

"

In Subclause 6.2.2.1, User data(), replace the user_data() syntax:
"

user_data() {	**No. of bits**	**Mnemonic**
user_data_start_code	32	bslbf
while(next_bits() != ‘0000 0000 0000 0000 0000 0001’) {		
user_data	8	uimsbf
}		
next_start_code()		
}		

"
with
"

user_data() {	**No. of bits**	**Mnemonic**
user_data_start_code	32	bslbf
while(next_bits() != ‘0000 0000 0000 0000 0000 0001’) {		
user_data	8	uimsbf
}		
}		

"

In Subclause 6.2.4, Group of Video Object Plane, in conformance to Table 6-3 replace the following row 2:
“

group_vop_start_codes	32	bslbf

"

with
“

group_of_vop_start_code	32	bslbf

"

In Subclause 6.2.5, Video Object Plane and Video Plane with Short Header, replace the following rows 18 to 26 of the VideoObjectPlane() syntax:

"

if(!(sprite_enable && vop_coding_type == “I”)) {		
vop_width	13	uimsbf
marker_bit	1	bslbf
vop_height	13	uimsbf
marker_bit	1	bslbf
vop_horizontal_mc_spatial_ref	13	simsbf
marker_bit	1	bslbf
vop_vertical_mc_spatial_ref	13	simsbf
}		

"

with

"

if(!(sprite_enable && vop_coding_type == “I”)) {		
vop_width	13	uimsbf
marker_bit	1	bslbf
vop_height	13	uimsbf
marker_bit	1	bslbf
vop_horizontal_mc_spatial_ref	13	simsbf
marker_bit	1	bslbf
vop_vertical_mc_spatial_ref	13	simsbf
marker_bit	1	bslbf
}		

"

In Subclause 6.2.5, replace the following rows 33 to 35 of the VideoObjectPlane() syntax:

“

}		
if (!complexity_estimation_disable)		
read_vop_complexity_estimation_header()		

"

with

“

}		
if (video_object_layer_shape != ‘’binary only‘’)		
if (!complexity_estimation_disable)		
read_vop_complexity_estimation_header()		

"

In Subclause 6.2.5, Video Object Plane and Video Plane with Short Header, replace row 44 of the VideoObjectPlane() syntax:

"

if (no_sprite_points > 0)		

"

with

"

if (no_of_sprite_warping_points > 0)		

"

In Subclause 6.2.5.2, Video Plane with Short Header, replace the video_packet_header() syntax:

"

video_packet_header() {	**No. of bits**	**Mnemonic**
next_resync_marker()		
resync_marker	17-23	uimsbf
macroblock_number	1-14	vlclbf
if (video_object_layer_shape != “binary only”)		
quant_scale	5	uimsbf
header_extension_code	1	bslbf
if (header_extension_code) {		
do {		
modulo_time_base	1	bslbf
} while (modulo_time_base != ‘0’)		
marker_bit	1	bslbf
vop_time_increment	1-16	bslbf
marker_bit	1	bslbf
vop_coding_type	2	uimsbf
if (video_object_layer_shape != “binary only”) {		
intra_dc_vlc_thr	3	uimsbf
if (vop_coding_type != “I”)		
vop_fcode_forward	3	uimsbf
if (vop_coding_type == “B”)		
vop_fcode_backward	3	uimsbf
}		
}		
}		

"

with

"

video_packet_header() {	**No. of bits**	**Mnemonic**
next_resync_marker()		
resync_marker	17-23	uimsbf
if (video_object_layer_shape != “rectangular”) {		
header_extension_code	1	bslbf
if (header_extension_code && !(sprite_enable && vop_coding_type == “I”)) {		
vop_width	13	uimsbf
marker_bit	1	bslbf
vop_height	13	uimsbf
marker_bit	1	bslbf
vop_horizontal_mc_spatial_ref	13	simsbf
marker_bit	1	bslbf
vop_vertical_mc_spatial_ref	13	simsbf
marker_bit	1	bslbf
}		
}		
macroblock_number	1-14	vlclbf
if (video_object_layer_shape != “binary only”)		
quant_scale	5	uimsbf
if (video_object_layer_shape == “rectangular”)		

header_extension_code	1	bslbf
if (header_extension_code) {		
do {		
modulo_time_base	1	bslbf
} while (modulo_time_base != ‘0’)		
marker_bit	1	bslbf
vop_time_increment	1-16	bslbf
marker_bit	1	bslbf
vop_coding_type	2	uimsbf
if (video_object_layer_shape != “rectangular”) {		
change_conv_ratio_disable	1	bslbf
if (vop_coding_type != “I”)		
vop_shape_coding_type	1	bslbf
}		
if (video_object_layer_shape != “binary only”) {		
intra_dc_vlc_thr	3	uimsbf
if (vop_coding_type != “I”)		
vop_fcode_forward	3	uimsbf
if (vop_coding_type == “B”)		
vop_fcode_backward	3	uimsbf
}		
}		
}		

"

In Subclause 6.2.5.3, Motion Shape Texture, replace the following rows 11, 12 and 13 of data_patitioned_i_vop() syntax:

"

if (!transparent_mb()) {		
mcbpc	1-9	vlclbf
if (mb_type == 4)		

"

with

"

If (!transparent_mb()) {		
if(video_object_layer_shape != “rectangle”){		
do{		
mcbpc	1-9	vlclbf
} while(derived_mb_type == “stuffing”)		
}else{		
mcbpc	1-9	vlclbf
if(derived_mb_type == “stuffing”)		
continue		
}		
if (mb_type == 4)		

"

In Subclause 6.2.5.3, Motion Shape Texture, replace the Note at the end of data_patitioned_i_vop() syntax:

"

NOTE The value of block_count is 6 in the 4:2:0 format. The value of alpha_block_count is 4.

"

with

"

NOTE 1 — The value of mb_in_video_packet is the number of macroblocks in a video packet. The count of stuffing macroblocks is not included in this value.
NOTE 2 — The value of block_count is 6 in the 4:2:0 format.
NOTE 3 — The value of alpha_block_count is 4.

"

In Subclause 6.2.5.3, Motion Shape Texture, replace the following rows 15 to 22 of the data_patitioned_p_vop() syntax:

"

if (!transparent_mb()) {		
not_coded	1	bslbf
if (!not_coded) {		
mcbpc	1-9	vlclbf
if (derived_mb_type < 3)		
motion_coding("forward", derived_mb_type)		
}		
}		

"

with

"

if (!transparent_mb()) {		
if(video_object_layer_shape != "rectangle"){		
do{		
not_coded	1	bslbf
if (!not_coded)		
mcbpc	1-9	vlclbf
} while(!(not_coded || derived_mb_type != "stuffing"))		
}else{		
not_coded	1	bslbf
if (!not_coded){		
mcbpc	1-9	vlclbf
if(derived_mb_type == "stuffing")		
continue		
}		
}		
if (!not_coded) {		
if (derived_mb_type < 3)		
motion_coding("forward", derived_mb_type)		
}		
}		

"

In Subclause 6.2.5.3, Motion Shape Texture, replace the Note at the end of data_patitioned_p_vop() syntax:

"

NOTE The value of block_count is 6 in the 4:2:0 format. The value of alpha_block_count is 4.

"

with

"

NOTE 1 — The value of mb_in_video_packet is the number of macroblocks in a video packet. The count of stuffing macroblocks is not included in this value. NOTE 2 — The value of block_count is 6 in the 4:2:0 format. NOTE 3 — The value of alpha_block_count is 4.

"

In Subclause 6.2.6, Macroblock, replace the following rows 6 to 11 of the macroblock() syntax:

"

if (!transparent_mb()) {		

if (vop_coding_type != "I" && !(sprite_enable
        && sprite_transmit_mode == "piece"))		

not_coded	1	bslbf
if (!not_coded || vop_coding_type == "I") {		
mcbpc	1-9	vlclbf

if (!short_video_header &&
            (derived_mb_type == 3 ||
            derived_mb_type == 4))		

"

with

"

if (!transparent_mb()) {		

if (video_object_layer_shape != "rectangular"
        && !(sprite_enable && low_latency_sprite_enable
        && sprite_transmit_mode == "update")) {		

do{		

if (vop_coding_type != "I" && !(sprite_enable
                && sprite_transmit_mode == "piece"))		

not_coded	1	bslbf

if (!not_coded || vop_coding_type == "I"
                || (vop_coding_type == "S"
                && low_latency_sprite_enable
                && sprite_transmit_mode == "piece"))		

mcbpc	1-9	vlclbf
} while(!(not_coded || derived_mb_type != "stuffing"))		
} else {		

if (vop_coding_type != "I" && !(sprite_enable
            && sprite_transmit_mode == "piece"))		

not_coded	1	bslbf

if (!not_coded || vop_coding_type == "I"
                || (vop_coding_type == "S"
                && low_latency_sprite_enable
                && sprite_transmit_mode == "piece"))		

mcbpc	1-9	vlclbf
}		

if (!not_coded || vop_coding_type == "I" || (vop_coding_type == "S" && low_latency_sprite_enable && sprite_transmit_mode == "piece")) {		
if (!short_video_header && (derived_mb_type == 3 || derived_mb_type == 4))		

"

In Subclause 6.2.8, Still Texture Object, replace the StillTextureObject() syntax:

"

StillTextureObject() {	**No. of bits**	**Mnemonic**
still_texture_object_start_code	32	
texture_object_id	16	uimsbf
marker_bit	1	bslbf
wavelet_filter_type	1	uimsbf
wavelet_download	1	uimsbf
wavelet_decomposition_levels	4	uimsbf
scan_direction	1	bslbf
start_code_enable	1	bslbf
texture_object_layer_shape	2	uimsbf
quantization_type	2	uimsbf
if (quantization_type == 2) {		
spatial_scalability_levels	4	uimsbf
if (spatial_scalability_levels != wavelet_decomposition_levels) {		
use_default_spatial_scalability	1	uimsbf
if (use_default_spatial_layer_size == 0)		
for (i=0; i<spatial_scalability_levels – 1; i++)		
wavelet_layer_index	4	
}		
}		
if (wavelet_download == "1"){		
uniform_wavelet_filter	1	uimsbf
if (uniform_wavelet_filter == "1")		
download_wavelet_filters()		
else		
for (I=0; i<wavelet_decomposition_levels; i++)		
download_wavelet_filters( )		
}		
wavelet_stuffing	3	uimsbf
if(texture_object_layer_shape == "00"){		
texture_object_layer_width	15	uimsbf
marker_bit	1	bslbf
texture_object_layer_height	15	uimsbf
marker_bit	1	bslbf
}		
else {		
horizontal_ref	15	imsbf
marker_bit	1	bslbf
vertical_ref	15	imsbf

marker_bit	1	bslbf
object_width	15	uimsbf
marker_bit	1	bslbf
object_height	15	uimsbf
marker_bit	1	bslbf
shape_object_decoding ( )		
}		
for (color = “y”, “u”, “v”)		
wavelet_dc_decode()		
if(quantization_type == 1)		
TextureLayerSQ ( )		
else if (quantization_type == 2){		
if (start_code_enable == 1) {		
do {		
TextureSpatialLayerMQ ( )		
} while (next_bits() == texture_spatial_layer_start_code)		
} else {		
for (i =0; i<spatial_scalability_levels; i++)		
TextureSpatialLayerMQNSC ( )		
}		
}		
else if (quantization_type == 3){		
for (color = “y”, “u”, “v”)		
do{		
quant_byte		
} while(quant_byte >>7)		
max_bitplanes		
if (scan_direction == 0) {		
do {		
TextureSNRLayerBQ ( )		
} while (next_bits() == texture_snr_layer_start_code)		
} else {		
do {		
TextureSpatialLayerBQ ( )		
} while (next_bits() == texture_spatial_layer_start_code)		
}		
}		
}		

"

with

"

StillTextureObject() {	**No. of bits**	**Mnemonic**
still_texture_object_start_code	32	
texture_object_id	16	uimsbf
marker_bit	1	bslbf
wavelet_filter_type	1	uimsbf
wavelet_download	1	uimsbf
wavelet_decomposition_levels	4	uimsbf
scan_direction	1	bslbf
start_code_enable	1	bslbf

texture_object_layer_shape	2	uimsbf
quantization_type	2	uimsbf
if (quantization_type == 2) {		
spatial_scalability_levels	4	uimsbf
if (spatial_scalability_levels != wavelet_decomposition_levels) {		
use_default_spatial_scalability	1	uimsbf
if (use_default_spatial_layer_size == 0)		
for (i=0; i<spatial_scalability_levels – 1; i++)		
wavelet_layer_index	4	
}		
}		
if (wavelet_download == “1”){		
uniform_wavelet_filter	1	uimsbf
if (uniform_wavelet_filter == “1”)		
download_wavelet_filters()		
else		
for (i=0; i<wavelet_decomposition_levels; i++)		
download_wavelet_filters( )		
}		
wavelet_stuffing	3	uimsbf
if(texture_object_layer_shape == “00”){		
texture_object_layer_width	15	uimsbf
marker_bit	1	bslbf
texture_object_layer_height	15	uimsbf
marker_bit	1	bslbf
}		
else {		
horizontal_ref	15	imsbf
marker_bit	1	bslbf
vertical_ref	15	imsbf
marker_bit	1	bslbf
object_width	15	uimsbf
marker_bit	1	bslbf
object_height	15	uimsbf
marker_bit	1	bslbf
shape_object_decoding ( )		
}		
for (color = “y”, “u”, “v”)		
wavelet_dc_decode()		
if(quantization_type == 1)		
TextureLayerSQ ( )		
else if (quantization_type == 2){		
if (start_code_enable == 1) {		
do {		
TextureSpatialLayerMQ ( )		
} while (next_bits() == texture_spatial_layer_start_code)		
} else {		
for (i =0; i<spatial_scalability_levels; i++)		
TextureSpatialLayerMQNSC ( )		
}		

}		
else if (quantization_type == 3){		
for (color = “y”, “u”, “v”)		
do{		
quant_byte		
} while(quant_byte >>7)		
max_bitplanes		
if (scan_direction == 0) {		
do {		
TextureSNRLayerBQ ( )		
} while (next_bits() == texture_snr_layer_start_code)		
} else {		
do {		
TextureSpatialLayerBQ ( )		
} while (next_bits() == texture_spatial_layer_start_code)		
}		
}		
}		

"

In Subclause 6.2.8.1, replace the TextureLayerSQ() syntax:

"

TextureLayerSQ() {	**No. of bits**	**Mnemonic**
if (scan_direction == 0) {		
for (“y”, “u”, “v”) {		
do {		
quant_byte	8	uimsbf
} while (quant_byte >> 7)		
for (i=0; i<wavelet_decomposition_levels; i++)		
if (i!=0 || color!= “u“,“v“) {		
max_bitplane[i]	5	uimsbf
if ((i+1)%4==0)		
marker_bit	1	bslbf
}		
}		
for (i = 0; i<tree_blocks; i++)		
for (color = “y”, “u”, “v”)		
arith_decode_highbands_td()		
} else {		
if (start_code_enable) {		
do {		
TextureSpatialLayerSQ()		
} while (next_bits() == texture_spatial_layer_start_code)		
} else {		
for (i = 0; i< wavelet_decomposition_levels; i++)		
TextureSpatialLayerSQNSC()		
}		
}		
}		

"