INTERNATIONAL STANDARD ISO/IEC 14496-3:2005/Amd.2:2006
TECHNICAL CORRIGENDUM 3

Published 2008-12-01

Information technology — Coding of audio-visual objects —

Part 3:
Audio

AMENDMENT 2: Audio Lossless Coding (ALS), new audio profiles and BSAC extensions

TECHNICAL CORRIGENDUM 3

Technologies de l'information — Codage des objets audiovisuels —

Partie 3: Codage audio

AMENDEMENT 2: Codage audio sans perte (ALS), nouveaux profils audio et extensions BSAC

RECTIFICATIF TECHNIQUE 3

Technical Corrigendum 3 to ISO/IEC 14496-3:2005/Amd.2:2006 was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee SC 29, *Coding of audio, picture, multimedia and hypermedia information*.

Throughout this Corrigendum, modifications or additions to existing text are highlighted in grey.

At the end of 11.2.1 Encoder and Decoder Structure, add the following sentence:

The decoder applies the inverse encoder operations in reverse order. Its output is a bit-identical version of the original input audio data.

ICS 35.040 **Ref. No. ISO/IEC 14496-3:2005/Amd.2:2006/Cor.3:2008(E)**

In 11.4.3, replace:

“nbit”

with:

“nbits[c][n]”

In 11.6.9.3.2.3, 11.6.9.3.2.4, and 11.6.9.3.2.5., replace all occurrences of:

nbit[c][n]

with

nbits[c][n]

At the end of 11.6.1.4, add the following paragraph:

If MCC (Multi Channel Coding) is selected, the relationship information between channels (master or slave) is decoded. The decoded residual values of the slave channel are modified by adding those of the master channel multiplied with the decoded weighting factors (see 11.6.8). Other reconstruction processes for all channel signals, which include parameter decoding, prediction residual decoding, synthesis filtering of long-term and short-term prediction, are identical to those for decoding independent channels. The two joint channel coding tools, joint-stereo and MCC, can be adaptively selected on a frame-by-frame basis.

In 11.6.3, replace:

If the prediction order K is adaptively chosen (adapt_order = 1), the number of bits used for signaling the actual order (opt_order = K) in each block is restricted, depending on both the global maximum order (max_order) and the block length N_B:

Bits = min{ceil[log2(max_order+1)], max[ceil(log2((N_B>>3)-1)), 1]}

Therefore, also the maximum order $K_{max} = 2^{Bits} - 1$ is restricted, depending on both the value of max_order and the block length (see Table 11.19).

Table 11.19 – Maximum prediction order depending on block length and max_order

N_B	max_order = 1023		max_order = 100	
	#Bits for opt_order	K_{max}	#Bits for opt_order	K_{max}
> 4096	10	1023	7	100
> 2048	9	511	7	100
> 1024	8	255	7	100
> 512	7	127	7	100
> 256	6	63	6	63
> 128	5	31	5	31
> 64	4	15	4	15
> 32	3	7	3	7
> 16	2	3	2	3
> 8	1	1	1	1

with:

If the prediction order K is adaptively chosen (adapt_order = 1), the number of bits used for signaling the actual order (opt_order = K) in each block is restricted, depending on both the global maximum order (max_order) and the block length N_B:

Bits = min{ceil[log2(max_order+1)], max[ceil(log2(N_B>>3)), 1]}

Therefore, also the maximum order K_{max} = min(2^{Bits} – 1, max_order) is restricted, depending on both the value of max_order and the block length (see Table 11.19).

Table 11.19 – Examples of maximum prediction orders depending on block length and max_order

N_B	max_order = 1023		max_order = 100	
	#Bits for opt_order	K_{max}	#Bits for opt_order	K_{max}
8192	10	1023	7	100
4096	9	511	7	100
2048	8	255	7	100
1024	7	127	7	100
512	6	63	6	63
256	5	31	5	31
128	4	15	4	15
64	3	7	3	7
32	2	3	2	3
16	1	1	1	1

In 11.6.3.1.2 , replace the first sentence:

First, Rice-decoded residual values δ_k are combined with offsets (see Table 11.20) to produce quantized indices of parcor coefficients a_k :

with:

First, the Rice-coded residual values δ_k are decoded and combined with offsets (see Table 11.20) to produce the quantized indices of the parcor coefficients a_k :

In 11.6.9.3.2.5, replace:

The algorithm for the Masked-LZ decompression is given below.

```
// Masked-LZ decompression.

long n, i, readBits, string_code;
unsigned long dec_chars

last_string_code = -1;
for ( dec_chars = 0; dec_chars < nchars; ) {
    readBits = inputCode( &string_code, code_bits );
    if ( string_code == FLUSH_CODE ) || ( string_code == MAX_CODE )) {
        FlushDict();
    }
    else if ( string_code == FREEZE_CODE ) {
        freeze_flag = 1;
    }
    else if ( string_code == bump_code ) {
        code_bits++;
        bump_code = bump_code * 2 + 1;
    }
    else {
        if ( string_code >= next_code ) {
            dec_chars += decodeStrint( &dec_buf[dec_chars], last_string_code, &charCode );
            dec_chars += decodeString( &dec_buf[dec_chars], charCode, &charCode );
            setNewEntryToDict( next_code, last_string_code, charCode );
            next_code ++;
        }
        else {
            dec_chars += decodeString(&dec_buf[dec_chars], last_string_code, &charCode);
            if ( ( dec_chars <= nchars ) && ( last_string_code != -1 ) && ( freeze_flag == 0 ) ) {
                setNewEntryToDict( next_code, last_string_code, charCode );
                next_code ++;
            }
        }
        last_string_code = string_code;
    }
}
```

Note: "dec_buf" is the buffer to store decoded characters. “nchars” is the number of characters need to be decoded. In FlushDict(), "code_bits" is set to 9, "bump_code" is set to 511 and "freeze_flag" is set to 0.
After input characters are decoded form code_bits, those characters are converted into difference values of the mantissa, D[c][n].

