Information technology — Coding of audio-visual objects — Part 3: Audio

TECHNICAL CORRIGENDUM 1

1 Corrections to ISO/IEC 14496-3:1999 (VERSION 1) subpart 5 (Structured Audio)

1.1 Encoding of class table_event unclear (editorial)

1.1.1 Problem:

In subclause 5.5.2, the class table_event encodes SASL table statements. The "which" value of the sample wavetable generator is encoded in a special optional section, controlled by the refers_to_sample flag bit. However, when coding this generator, a place should also be left in the pf[] array for "which" (the value placed in this position is ignored), so that the optional parameter "skip" occurs in the proper place in pf[].

1.1.2 Correction:

In subclause 5.5.2, include comment under pf[] array in table_event

" // when coding sample generator, leave a blank array slot
  // for "which" parameter, to maintain alignment for "skip" parameter "

1.2 Bitstream syntax for concat table generator (technical)

1.2.1 Problem:

In subclause 5.5.2 the definition of class table_event does not support the case of "concat" table generator, where many tables are possible as arguments.

1.2.2 Correction:

Replace the line

```
float(32) pf[num_pf] ;
```

with the following lines:

```
if (tgen==0x7D) {  // concat
   float(32) size;
   symbol ft[num_pf - 1];
} else {
   float (32) pf[num_pf] ;
}
```

1.3 Zero-length strings in symbol table (editorial)

1.3.1 Problem:

Subclause 5.5.2, describing the symbol table, doesn't state a class sym_name with a zero-length string encodes the fact that a symbol number does not have a symbol name.

1.3.2 Correction:

In subclause 5.5.2, second paragraph, append this sentence at the end of paragraph:

"The presence of a zero-length string in a symbol table entry indicates that a name for this symbol is not included in the symbol table".

1.4 Decoder execution while streaming is not precise (technical)

1.4.1 Problem:

In subclause 5.7.3.3.6 points 1 to 7 there are comparisons with the orchestra absolute time to execute events or modify them. Using the words "earlier than" implies that an event programmed to be precisely executed at an instant is instead delayed until the next kcycle, which could be several milliseconds after the correct time. This can also cause serious problems to initialize control parameters at the instant t=0.

1.4.2 Correction:

Replace "earlier than" in the first line of points 1 to 7 with "earlier than or equal to"

1.5 Precedence of expression operators (technical)

1.5.1 Problem:

In subclause 5.8.6.7.14 the order of operations slightly differs from the order that is standardized for the C programming language. This is in some way misleading and could lead to confusion among programmers; moreover Table 5.3 does not correspond to syntactic grammar reported in Annex 5.C (highest precedences are inverted). It is proposed to adopt the operator precedence of the C language.

1.5.2 Correction:

In Table 5.3 group unary negation and not on a single line. Replace the last line of text in subclause 5.8.6.7.14 with the following text: "Operations listed on the first and last row associate right-to-left, that is, the rightmost expression is performed first. Operations listed on the remaining rows associate left-to-right, that is, the leftmost expression is performed first.
In Annex 5.C.3 the paragraph starting with "%start orcfile" is replaced by the following one:

```
%start orcfile
%right Q
%left OR
%left AND
%left EQEQ NEQ
%left LT GT LEQ NEQ
%left PLUS MINUS
%left STAR SLASH
%right UNOT UMINUS
%token HIGHEST
```

1.6 SAOL instr command temporal parameters units unclear (editorial).

1.6.1 Problem:

In subclause 5.8.6.6.7 (the SAOL instr command), the units of duration and delay parameters may be misunderstood, and should be stated directly to foster interoperable decoder implementations.

1.6.2 Correction:

To paragraph six in subclause 5.8.6.6.7, add the line

"To clarify, note that the values of the first and second expressions are considered to have units of score beats, not absolute time, and they are consequently scaled according to the actual **tempo** of the orchestra"

1.7 MIDI standard name visibility in dynamic instrs (technical)

1.7.1 Problem:

If an instrument instance is instantiated via a MIDI NoteOn event, as described in subclause 5.14.3.2.3, and this instance uses the SAOL instr statement, as described in subclause 5.8.6.6.7, the standard is silent on the visibility of the MIDI standard names in the dynamically created instrument.

1.7.2 Correction:

In subclause 5.8.6.6.7, add the following paragraph at the end:

"A dynamically created instrument has access to the same **MIDIctrl** (subclause 5.8.6.8.9), **MIDItouch** (subclause 5.8.6.8.10), **MIDIbend** (subclause 5.8.6.8.11), **channel** (subclause 5.8.6.8.12) and **preset** (subclause 5.8.6.8.13) standard name state as its parent. However, the dynamically created instrument is not scheduled for termination when the parent is terminated under MIDI control.

1.8 params[] standard name not listed as an lvalue (editorial)

1.8.1 Problem:

The params[] standard name (subclause 5.8.6.8.26) is meant for bi-directional communications in BIFS, but this standard name is not permitted to be an lvalue (subclause 5.8.6.6.2).

1.8.2 Correction:

Copy paragraph 2 of subclause 5.8.6.8.9 (which describes how the **MIDIctrl**[] standard name may be used as an lvalue) to the **params[]** subclause 5.8.6.8.26, changing all references of **MIDIctrl[]** to **params[]**. Also change subclause 5.8.6.6.2 to permit the usage of **params[]** as an assignment lvalue.

1.9 Outbus statement not precluded from writing to input_bus (editorial)

1.9.1 Problem:

In subclause 5.8.6.6.10, the outbus statement is defined. No mention is made of the legality of writing to the special input_bus.

1.9.2 Correction:

Add to the end of subclause 5.8.6.6.10:

"In addition, an **outbus** statement may not write to the special bus **input_bus**".

1.10 lvalue standard names not permitted as call-by-reference opcode arguments (technical)

1.10.1 Problem:

The **MIDIctrl[]** (subclause 5.8.6.8.9) and **params[]** (subclause 5.8.6.8.26, with Corrigenda 1.21 semantics) may be used as an lvalue in an assignment statement, so programs may change their value. Programs may not, however, change the value of these standard names by using call-by-reference via an opcode or oparray call, because the prohibition against standard names in call-by-reference in subclause 5.8.6.7.6 does not make exceptions for lvalue standard names.

1.10.2 Correction:

In the second to last paragraph of subclause 5.8.6.7.6 (not including NOTES), replace the phrase "unless the parameter is a standard name" with "unless the parameter is a standard name which may not be used as an lvalue".

1.11 Statements inside a user-defined opcode (technical)

1.11.1 Problem:

In subclause 5.8.7 it is not specified nor clear if and how statements that are slower than the user-defined opcode are executed inside the opcode itself. This may lead to important differences of behavior among the different decoders and the problem must be fixed. It seems reasonable to adopt rules similar to those that can be derived from core opcodes. Moreover, according to the rules in subclause 5.8.8.7.2, variables and statements in a rate-polymorphic opcode faster than the fastest formal parameter could result in an unpredictable result. It seems wise to explicitely forbid this case.

1.11.2 Correction:

In subclause 5.8.7.6.1, change the end of the second paragraph of text as follows:
"No statement in an opcode shall be faster than the rate of the opcode, as defined in subclause 5.8.7.7. A statement that is slower than the rate of the opcode shall be executed as described in subclause 5.8.7.7.3"

In subclause 5.8.7.7.2 add the following statement before the example.

"Rate-polymorphic opcodes shall not contain variable declarations and statements faster than the fastest formal parameter in the opcode declaration. In particular an opcode with all **xsig** formal parameters shall not contain variable declarations other than **xsig** and **ivar** and it shall not contain statements at a defined rate faster than the initialization rate."

Add a subclause 5.8.7.7.3 as follows:

"
5.8.7.7.3 Shared variables and statements slower than the rate of the opcode

This subclause describes the rules for updating shared variables and for executing statements inside an opcode that are slower than the rate of the opcode.

In a **kopcode**, statements at the initialization rate are executed at the first call to the opcode with regard to a particular opcode state. At this call the **imports**, **exports** and **imports exports** ivars and tables are updated, as described in 5.8.6.5.3 and 5.8.6.5.4, and any wavetable generations are executed, as described in 5.8.6.5.2.

In an **aopcode**, statements at the initialization rate are executed at the first call to the opcode with regard to a particular opcode state. At this call the **imports**, **exports** and **imports exports** ivars and tables are updated, as described in 5.8.6.5.3 and 5.8.6.5.4, and any wavetable generations are executed, as described in 5.8.6.5.2.
Statements at the control rate are executed at the first call of every k-cycle to the opcode with regard to a particular opcode state. At this call the **imports**, **exports** and **imports exports** ksigs and tables are updated, as described in 5.8.6.5.3 and 5.8.6.5.4

In an **opcode**, once the rate is determined as specified in subclause 5.8.7.7.2 the same rules apply for the cases of call at **k-rate** or **a-rate** respectively."

Add in the last paragraph of subclause 5.8.6.7.6, concerning call by reference semantics:

"If the rate of a formal parameter is slower than the rate of the opcode, the following rules apply:

in an opcode call at the **k-rate**, actual variables associated to an **ivar** formal parameters are updated only the first time this opcode is executed with regards to a particular opcode state;

in an opcode call at the **a-rate**, actual variables associated to an **ivar** formal parameters are updated only the first time this opcode is executed with regards to a particular opcode state; actual variables associated to a **ksig** formal parameters are updated only the first time this opcode is executed in a k-cycle with regards to a particular opcode state."

1.12 opcode/oparray call rate semantics clarification (technical)

1.12.1 Problem:

In subclause 5.8.6.7.6, paragraph 7, some uses of opcode calls are disallowed on a rate-semantics basis. Specifically, opcode calls in guarded blocks may not have a rate slower than the guard. Apart from this reference, however, no rate-semantics prohibitions exist for opcode calls, and so, for example:

```
kopcode unity_krand() { return krand(1); }

instr test ()

   {

     asig a;

     a = unity_krand();

   }
```

is a legal code fragment, where the assignment statement in test() executes at the a-rate. However, there is no explicit guidance in subclause 5.8.6.7.6 or elsewhere with regard to the rate semantics of executing a kopcode or iopcode at the a-rate, or executing an iopcode at the k-rate. This situation is equally true for oparray calls as defined in subclause 5.8.6.7.7. To ensure normative decoder implementations, explicit guidance is needed.

1.12.2 Correction:

Add these paragraphs after paragraph 7 in subclause 5.8.6.7.6:

"If a **kopcode** opcode call occurs in a expression that runs at the **a-rate**, the first time this expression is executed in a k-cycle with regards to a particular opcode state, the **kopcode** is called, following the semantics described in this subclause. For all subsequent evaluations of the expression in the same k-cycle, the **kopcode** is not executed; instead, the return value from the first execution is used in the expression evaluation.

If an **iopcode** opcode call occurs in a expression that runs at the **a-rate** or the **k-rate**, the first time this expression is executed with regards to a particular opcode state, the **iopcode** is called, following the semantics described in this subclause. For all subsequent evaluations of the expression, the **iopcode** is not executed; instead, the return value from the first execution is used in the expression evaluation.

If a **specialop** core opcode call occurs in a expression that runs at the **a-rate**, the **k-rate** semantics of the **specialop** opcode call follows the rules for **kopcode** calls described above, while the **a-rate** semantics of the **specialop** opcode call happen at every a-cycle as described in subsection 5.9.2."

For subclause 5.8.6.7.7, change paragraph 6 as follows:

"The context of the **oparray** call expression is restricted in the same way as described for the **opcode** call expression in subclause 5.8.6.7.6. The rate semantics for **oparray** call execution follows the same rules described for the **opcode** call expression in subclause 5.8.6.7.6."

1.13 Legal if and if-else constructions have under-specified semantics (technical)

1.13.1 Problem:

The rate-semantic restrictions in subclause 5.8.6.4 and 5.8.6.5 permit the following code fragment, but do not specify the semantics it should execute:

```
ivar i;

  asig a;

  ksig k;

if (irand(1) < 0.5)

   {

     i = i + 1;

     k = k + i;

     a = k;

   }
```

The if statement executes at the a-rate, if the guard condition is met, but the scheduling of the i-rate and k-rate assignment statements is left unspecified. Different decoders may implement different semantics in these cases, with non-normative consequences.

1.13.2 Correction:

Add these paragraphs to subclauses 5.8.6.6.4 and include it by reference in 5.8.6.6.5:

"If a block of code executing at the **a-rate** or **k-rate** has **i-rate** statements, these statements should only be executed the first time the block executes, with regards to a particular state.

If a block of code executing at the **a-rate** has **k-rate** statements, these statements should only be executed the first time the block executes in a kcycle."

1.14 return statement width semantics unclear (technical)

1.14.1 Problem:

In subclause 5.8.7.6.2, rules for determining the width of a user-defined opcode are presented, in the context of describing the return statement. However, no mention is made of the proper width semantics if no return statement is provided, or if the return statement has an expression list of width zero (a legal syntax in subclause 5.8.7.6.1)

1.14.2 Correction:

In subclause 5.8.7.6.2, add this line at the end of paragraph 4:

"If an opcode has a return statements with an <expr list> of length 0, it is considered to have width 1 for the purposes of this paragraph. If the opcode executes this **return** statement, the return value is undefined.

If an opcode has no **return** statements, it has a width of 1, and its return value is undefined. If no **return** statement is encountered during an opcode execution, control is returned to the calling instrument or opcode at the end of the statement block."

1.15 maximum value in dbamp and ampdb is not 1

1.15.1 Problem:

In subclauses 5.9.4.4 and 5.9.4.5 it is stated that 1 is the maximum value for amplitude. This is not constrained by run-time errors; moreover, the input/output range of a SAOL orchestra shall be in the range -1 to 1 but this does not apply to internal arithmetic, which has the only constraint of the float32 compatibility.

1.15.2 Correction

Remove the word "maximum" from subclauses 5.9.4.4 and 5.9.4.5

1.16 Negative frequency behavior in core opcodes unspecified (technical)

1.16.1 Problem:

Many core opcodes include a formal parameter for frequency, which in many cases controls the rate a phase pointer cycles through a table. Several important sound generation algorithms depend on the concept of negative frequency, such as FM.

None of the core opcode semantic descriptions explicitly mention that negative frequencies are permissible, and as a result decoders may not implement the negative frequency concept correctly.

1.16.2 Correction:

In subclauses:

5.9.6.12 (oscil core opcode)

5.9.6.13 (loscil core opcode)

5.9.6.15 (koscil core opcode)

5.9.7.5 (kphasor core opcode)

5.9.7.6 (aphasor core opcode)

5.9.7.7 (pluck core opcode)

5.9.7.8 (buzz core opcode)

add the following paragraph at the end of each description:

"Note that a negative value for the frequency argument to this opcode is legal, and the phase pointer update algorithm works correctly for negative values."

In addition, for subclause 5.9.6.13 (loscil core opcode), add the following text to the end of the paragraph that begins with "Let l be the length of the table ..."

"However, if the phase pointer has not yet passed the **loopstart** value, and if an incrementation has caused the phase pointer to become less than zero, the phase pointer takes on the value zero."

1.17 Acceptable range in the tablewrite core opcode (technical)

1.17.1 Problem:

In subclause 5.9.6.11 **index** cannot be set to the size of the table, since acceptable indexes ranges from 0 to size-1 and the value is rounded to the nearest integer.

1.17.2 Correction:

Replace the second sentence as follows:

"It is a run-time error if **index** < 0, or if **index** is greater than or equal to (tsize-0.5), where tsize is the size of the wavetable referenced by **t**."

1.18 aline core opcode definition (editorial)

1.18.1 Problem:

In subclause 5.9.7.2 the declaration of the aline core opcode as kopcode is wrong, as evident from the subsequent description. Moreover the x in the equation of the return value is misleading.

1.18.2 Correction:

Change the definition to aopcode.

Remove the x character from the equation of the returned value ($l + (r - l)t/d$)

1.19 buzz core opcode (technical)

1.19.1 Problem:

In subclause 5.9.7.8 **scale** equation for (**rolloff**!=1) is slightly incorrect. The cos(..f.) under summation needs to be cos(..(f+1).) in order not to obtain a DC component with lowharm == 0.
If **rolloff** == 1, **scale** must be explicitely defined to avoid discontinuity.

1.19.2 Correction:

Correct the return value equation.

If |**rolloff** = 1|, **scale** = (1/(nharm+1))

else **scale** = (1-abs(**rolloff**))/(1-abs(**rolloff**$^{(\text{nharm} + 1)}$))

1.20 Use of amp parameter in grain core opcode unspecified (editorial)

1.20.1 Problem:

In subclause 5.9.7.9, the formal parameter list of the grain core opcode includes the ksig amp parameter, but the semantic description of the opcode does not specify its use.

1.20.2 Correction:

Change the last line of the second-to-last paragraph of subclause 5.9.7.9 (not including NOTES) to be:

"The final output value of the grain **x[i]** is rescaled by this modulator value as in **x[i]** = **amp*x[i]*m[i]**."

1.21 Error in the definition of the biquad core opcode functionality (technical)

1.21.1 Problem:

In subclause 5.9.9.6 the pseudo-code that defines the opcode functionality does not correspond to the biquad equation.

1.21.2 Correction:

Replace the pseudo-code with the following one:

```
ti = input;
to = w2 + b0*ti;
w2 = w1 - a1*to + b1*ti;
w1 = -a2*to + b2*ti;
```

Delete w0 from the list of intermediate variables.

1.22 allpass and comb core opcode delay line lengths underspecified (editorial)

1.22.1 Problem:

Many core opcodes define internal delay lines, whose length is specified by a user parameter, in units of seconds. The descriptions of the core opcodes supply an equation for converting seconds to delay line length that uses the floor() function to ensure normative rounding. The allpass and comb core opcodes do not specify delay line length in this formal way.

1.22.2 Correction:

In subclause 5.9.9.7 and 5.9.9.8, paragraph 2, replace the expression "**time***SR" with floor(**time***SR).

1.23 Paired usage of fft and ifft opcodes produces incorrect results in shift-add mode (technical)

1.23.1 Problem:

Subclause 5.9.10.1 and 5.9.10.2 define a pair of spectral opcodes (fft() and ifft()) that are designed to be used together for sound synthesis. Several errors in the definition of the opcodes results in errors when shift-add mode is used with these opcodes. Errors take the form of incorrect amplitude scaling, and evident waveform distortion during the first several opcode calls.

1.23.2 Correction:

In subclause 5.9.10.1, replace the fourth paragraph with the following paragraph:

"The calculation of this opcode is as follows. On the first call to the **fft** opcode with respect to a particular state, a holding buffer of length **len** is created, and the first (**len** - **shift**) buffer samples are initialized with zeros. On each a-rate call to the opcode, the **input** sample is inserted into the next uninitialized buffer sample position. When all **len** samples in the buffer are initialized, the following steps are performed:"

In addition, in the display equation following

"The DFT is defined as:"

replace all uses of "len" with "size", including the variable descriptions under the equation. Also replace x[k] with new[k] in both equation and variable description.

In subclause 5.9.10.2, at point 2 replace

(**out[i] += seq[i]*win[i]** for 0<**i**<**len**)

with:

(**out[i] += seq[i]*win[i]** ***shift**/**len** for 0<**i**<**len**)

In addition, in the display equation following

"The IDFT is defined as:"

replace all uses of "len" with "size", including the variable descriptions under the equation. Also replace x[k] with seq[k] in both equation and variable description, and in the variable description replace "input samples" with "output samples".

1.24 Example showing usage of fft() and ifft() opcodes does not produce claimed behavior (editorial)

1.24.1 Problem:

The example in 5.9.10.2 showing correct fft()/ifft() usage has errors which may confuse implementors. In particular, in the example as it stands, modifying the (re,im) tables produced by fft() will not result in ifft() generating audio from the modified tables, as fft() will overwrite the modified tables before ifft() can turn them into audio.

1.24.2 Correction:

[1] The "table t(empty, 1025)" declaration needs to be replaced with

```
table re_original(empty, 1024);

table im_original(empty, 1024);

table re_modified(empty, 1024);

table im_modified(empty, 1024);
```

[2] The fft() call should use the re_original and im_original tables.

[3] The ifft() call should use the re_modified and im_modified tables.

[4] Comment next to the "if(new_fft)" fragment should be:

```
// read original tables, generate modified values,

// and place these values into modified tables.
```

1.25 Declaration of formal parameters in the compressor core opcode (editorial)

1.25.1 Problem:

In subclause 5.9.11.4 the formal parameter `look` is declared as "`isig`".

1.25.2 Correction:

Replace `isig` with `ivar`.

1.26 Wrong definition of gain, incorrect semantics for ratio, incorrect figure and noise gate definition in the compressor core opcode (technical-editorial)

1.26.1 Problem:

In subclause 5.9.11.4 the description at number 3 does not correspond to the behavior of a compressor and to Figure 5.3. Figure 5.3 is not clear and its modeling of noise shaping is wrong. The limited opcode dynamic range of

90 dB is in clear contrast with the numeric format of SA. Some text is also misleading at point 3. Some editorial changes are necessary for conformity and clarity, like parentheses for guard expressions and braces.

1.26.2 Correction:

Several corrections/clarifications are necessary:

a) add a new formal parameter to the definition of compressor; between comp and thresh insert "ksig nfloor"

b) replace "**thresh** is negative" in the list of run-time errors with "**thresh** < **nfloor**". Replace the "**ratio** is 0" in the same list with "**ratio** is equal to or less than 0".

Note that in such way the semantics for ratio values less than zero in the original opcode definition are no longer a part of the specification.

c) after the att and rel paragraph add the following:

"**nfloor** will set the absolute decibel floor for the system. It defines the minimum value to which the compressor is potentially reactive. For instance, if **nfloor** is 0, according to the decibel scale of SAOL, the minimum recognized absolute value for input will be $10^{-90/20}$, if **nfloor** is -10 the minimum recognized absolute value will be $10^{-(90+10)/20}$, and so on"

d) replace the thresh paragraph with the following:

"**thresh** is the minimum decibel level that will be allowed through. For a noise gate, this value will be greater than **nfloor**."

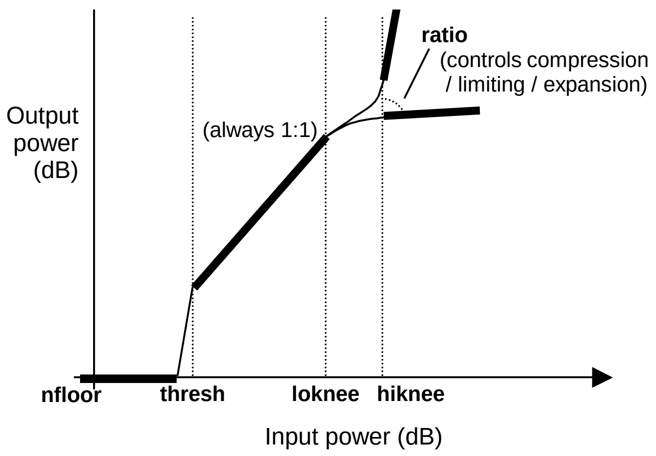
e) define ratio as:

"**ratio**, given in dB, defines compression above the knee. It defines the change in input power required for a 1dB change in output power. **ratio** values above 1 dB result in compression, with higher values yielding greater compression. Numbers between 0 and 1 result in expansion. For example, a **ratio** value of 3dB implies a compression (a 3dB change in input power produces a 1dB change in output power) while a **ratio** value of 0.5dB implies an expansion (a 0.5dB change in input power produces a 1dB change in output power)"

f) add the following paragraph before the figure

"The compression opcode parameters **nfloor**, **thresh**, **loknee**, and **hiknee** are defined on the decibel scale defined in subclause 5.9.4.4 for the **dbamp** core opcode. In this space, a waveform value of 1.0 maps to a decibel value of 90dB, a waveform value of 0.1 maps to 70dB, etc. Note that negative decibel values are permitted."

g) replace current Figure 5.3 with the following one:



h) replace the first of the examples following the figure (paragraph starting with "The actions of **compressor**()) with the following:

"a hard-knee compressor is obtained from equal-value **loknee** and **hiknee** break-points";

i) replace the four points as follows:

1. The sample **x** is placed into the beginning of the buffer **xdly**, pushing all values down and the oldest value off the end. The value pushed off the end is saved as **oldval**.

2. The next envelope value is calculated as follows:

   abs(**comp**) is converted to decibels. This value is called **comp1** with

   comp1 = 90 + 20*log10(abs(**comp**)).

   It is put into the end of the buffer **compdly**. The value **comp2** is taken from the beginning of the buffer **compdly**.

```
if (comp2 > env)

     change = (comp2 – env) / (SR * att)

else {

    projEnv = max(change * SR * look, nfloor)

    if ((comp1 > projEnv) && (comp2 > comp1))

         change = (comp1 – comp2) / (SR * rel)

    else

         change = 0;

}

env = max(env + change, nfloor);
```

3. The amplitude multiplier **gain** is calculated as follows:

```
if (env < thresh)

     gain shall first decrease monotonically from 1 to 0 and then stay at 0, moving from thresh towards
     nfloor, to create a noise gate. The exact input-output curve in this interval is not specified; it is left to
     the implementation. Implementations may also include temporal behavior in this regime to reduce
     onset and offset signal distortion.

else {

     if (env <= loknee)

          gain = 1

     else if (env >= hiknee)

          gain shall be calculated so that, above hiknee, an increase of ratio dB in input power must
          result in an increase of 1dB in output power. The decibel input/output curve shall be continuous
          at hiknee with either the soft or the hard knee curve.

     else

          gain shall be interpolated smoothly between the loknee and hiknee points to create a soft-knee
          input-output (decibel) curve. This curve shall be monotonically increasing, and have continuous
          derivative equal to 1 at loknee and 1/ratio at hiknee. The exact input-output curve in this interval
          is not specified; it is left to the implementation.
```

}

4. The output value of the opcode shall be **oldval** multiplied by **gain**.

1.27 Percentage definition of depth in chorus and flanger core opcodes (technical)

1.27.1 Problem:

In subclauses 5.9.14.1 and 5.9.14.2 **depth** is specified as percent excursion, but its range is not specified.

1.27.2 Correction:

Add the following text:

"depth is specified as percent excursion with 0<=**depth**<=100. It is a run-time error if **depth**<0 or **depth**>100"

1.28 Expseg wavetable generator (editorial)

1.28.1 Problem:

In subclause 5.10.7 the equation for range x2 through x3 manifestely contains a "copy-and-paste" error.

1.28.2 Correction:

Replace line:

x in the range **x2** through **x3** shall be set to $\mathbf{y2}(\mathbf{y3}/\mathbf{y2})^{(x-\mathbf{x1})/(\mathbf{x2}-\mathbf{x1})}$

with line:

x in the range **x2** through **x3** shall be set to $\mathbf{y2}(\mathbf{y3}/\mathbf{y2})^{(x-\mathbf{x2})/(\mathbf{x3}-\mathbf{x2})}$

1.29 spline wavetable generator parameter run-time error check incorrect (technical)

1.29.1 Problem:

Subclause 5.10.9 states that

"it is a run-time error if:

[...]

there are a odd number of parameters, not counting the size parameter."

This statement is incorrect, as this wavetable generator requires 2 + N*3 parameters, a number that may be even or odd depending on N.

1.29.2 Correction:

Change the error listing in subclause 5.10.9 to delete:

"there are a odd number of parameters, not counting the **size** parameter."

And add:

"there are less than 4 parameters, not including **size**" "the last parameter is not a k value"

1.30 Gaussian window width in window wavetable generator (technical)

1.30.1 Problem:

In subclause 5.10.11 the equation for gaussian generation produce a too narrow window to be practically useful. A new proposed equation produces more suitable values.

1.30.2 Correction:

Replace the paragraph for type == 4 as follows:

If **type**==4, a Gaussian window shall be used. For sample number x in the range [0, **size**-1], the value placed in the table shall be

$e^{-c1(c2-x)(c2-x)}$, where c2 = **size**/2 and c1 = 18/(**size**)^2

1.31 Nondeterministic wavetable creation order clashes with concat wavetable generator semantics (technical)

1.31.1 Problem:

The concat wavetable generator (subclause 5.10.16) combines multiple tables into a new table via concatenation. The use of this generator relies on its input wavetables being created before it is called. However, subclause 5.8.5.3.3 on wavetables, paragraph 2 states:

"The order of creation of wavetables is not deterministic;"

1.31.2 Correction:

Change subclause 5.8.5.3.3 line quoted above to state:

"The order of creation of wavetables is not deterministic, with the exception of the table arguments of a **concat** generator, which are always generated before the **concat** generator that uses them. In this case, the tables used as arguments to the **concat** generator must be appear before the table which uses the **concat** generator, to prevent dependency loops."

1.32 MIDI All Sound Off and All Notes Off event command issues (technical)

1.32.1 Problem:

Subclause 5.14.3.2.11 specified normative behavior in response to the MIDI All Notes Off event, but its behavioral description in this subclause matches the MIDI All Sound Off command. Meanwhile, the MIDI All Sound Off is missing from the description of MIDI Event semantics (subclause 5.14.3.2).

1.32.2 Correction:

Change subclause 5.14.3.2.11, and add a new subclause 5.14.3.2.14, as follows:

"

5.14.3.2.11 All notes off

When a **notesoff** event is received, all instrument instances in the orchestra created by a MIDI NoteOn events (subclause 5.14.3.2.3) are scheduled for termination at the end of the current k-cycle; that is, the released flag is set, and if the instrument does not call extend, it shall be de-instantiated after the current k-cycle of computation.

If the **MIDIctrl**[64] value for an instance is non-zero, then the execution of the termination shall be delayed until the **MIDIctrl**[64] value becomes zero.

These semantics correspond to the behavioral intentions of the MIDI All Notes Off command: the release cycle of notes are permitted to decay naturally, and sustain pedal semantics are obeyed. The **MIDIctrl**[123] value, normally 0, shall be set to 1 for all associated MIDIctrl accesses during the orchestra cycle the All Notes Off Command is processed, so that dynamic instruments spawned from the MIDI instrument may detect the All Notes Off command.

5.14.3.2.14 All sound off

When a **soundoff** event is received, all instrument instances in the orchestra created by MIDI NoteOn events (subclause 5.14.3.2.3) are terminated at the end of the current k-cycle. Instruments may not save themselves from termination using the extend statement in this case, and the status of the **MIDIctrl**[64] is irrelevant. The **MIDIctrl**[120] value, normally 0, shall be set to 1 for all associated **MIDIctrl** accesses during the orchestra cycle the All Sound Off Command is processed, so that dynamic instruments spawned from the MIDI instrument may detect the All Sound Off command.

These semantics correspond to the behavioral intentions of the MIDI All Sound Off command: an immediate ending of all sound making.

"

1.33 Effects instruments cannot monitor MIDI standard names (technical)

1.33.1 Problem:

Programmers wish to write SA programs that act as real-time audio signal processing systems. The core instruments of the program are instantiated using the SAOL send command, with the first instruments in the processing chain being sent the special input_bus to receive external audio input.

Users wish to control these signal processing systems using MIDI controllers, via the MIDIctrl[], MIDIbend, and MIDIwheel standard names. However, the standard makes no mention of the use of these standard names in instruments not instantiated via MIDI NoteOn commands.

1.33.2 Correction:

Add a new section 5.14.3.2.15

5.14.3.2.15 The MIDI Master Channel

All instances created by SASL instr statements and SAOL send statements, and any dynamic instruments instantiated from instances created by SASL instr statements and SAOL send statements, see the status of the MIDI Master Channel in the values of the **MIDIctrl**[], **MIDIbend**, **MIDIwheel**, and **channel**, and **preset** standard names. For these instances, the value of the **channel** standard name reflects the extended channel number of the MIDI master channel, and the value of the **preset** standard name reflects the value of the last pchange event on the MIDI Master Channel.The values of the **MIDIctrl**[], **MIDIbend**, **MIDIwheel** are identical to the values seen in an instance instantiated via a MIDI NoteOn event from the MIDI Master Channel.

The identity of the MIDI Master Channel is determined as follows:

If a MIDI source is directly connected to the orchestra, as described in Annex 5.F, then the MIDI Master Channel is MIDI channel 0 of this source.

If no MIDI source is directly connected to the orchestra, if an SA_access_unit MIDI source is present, the MIDI Master Channel is channel 0 of this source.

If no MIDI source is directly connected to the orchestra, and if no SA_access_unit MIDI source is present, the MIDI Master Channel is channel 0 of the first MIDI File track that contains a NoteOff, NoteOn, CChange, PChange, Pwheel, Touch, or CTouch command on channel 0.

Decoder implementations may provide a way to specify the MIDI Master Channel directly, superseding these rules, if the decoder permits the direct connection of MIDI sources to the orchestra.

1.34 MIDI File default tempo is the double than the SAOL default tempo (compatibility)

1.34.1 Problem:

The MIDI File standard encourages authors to set a default tempo value in the file, but states that if the tempo is not specified in the MIDI file, it should default to 120 beats per minute. In contrast, the default SAOL tempo is 60 beats per minute (subclause 5.9.15.1). As a result, some MIDI files play twice as slow, unless a SASL tempo command or SAOL settempo() opcode is used to compensate.

1.34.2 Correction:

Add this line to 5.14.3.3.3, at the end of paragraph 1:

"If the MIDI file does not specify a starting **tempo**, the default value of 120 beats per minute shall be used."

1.35 phaseGroup usage misleading in subclause 5.15 (editorial)

1.35.1 Problem:

Subclause 5.15.2 describes the use of phaseGroup in depth. However, description does not clarify that phaseGroup is only used outside the orchestra scope, and this could lead to misunderstanding.

1.35.2 Correction:

Add to subclause 5.15.2 the following NOTE

"The **phaseGroup** values are made available to the SA decoder, but they are not visible inside the scope of the several orchestra elements, and therefore **phaseGroup** does not constitute a standard name as **inGroup**. **phaseGroup** is only used outside the orchestra by the SA decoder to construct the **inGroup** values."

1.36 Errors in the bitstream token table (technical)

1.36.1 Problem:

In Annex 5.A the startup token is missing in the bitstream token table. The destroy token is instead not needed.

1.36.2 Correction:

Add the startup token as 0x4B (from reference software)

0x7F becomes **(reserved)**

1.37 Yacc grammar clarification with regards to templates (editorial)

1.37.1 Problem:

Appendix 5.C.3 (a yacc grammar for SAOL) does not describe the template syntax presented in subclause 5.8.8.2.

1.37.2 Correction:

Add this text to the bottom of the comment block at the start of Appendix 5.C.3:

"Note that this grammar does not fully support the template construct as defined in subclause 5.8.8.2. This subclause permits expressions in the preset and map lists, whereas this grammar only supports terminals in these lists. To support the full template syntax, a lexical analyzer that detects expressions in preset and map lists and takes special action is necessary."

1.38 Yacc grammar clarification with regards to user-defined opcodes and tablemaps (editorial)

1.38.1 Problem:

The yacc grammar in Appendix 5.C.3 does not let user-defined opcodes declare tablemaps (see the opvardecl definition) but subclause 5.8.7.5.1 permits tablemap declarations in user-defined opcodes.

1.38.2 Correction:

In Appendix 5.C.3, copy the TABLEMAP line of the vardecl definition to the opvardecl definition.