INTERNATIONAL STANDARD

ISO/IEC 23000-12

First edition
2010-07-15

AMENDMENT 2
2012-05-01

Information technology — Multimedia application format (MPEG-A) —

Part 12: Interactive music application format

AMENDMENT 2: Compact representation of dynamic volume change and audio equalization

Technologies de l'information — Format pour application multimédia (MPEG-A) —

Partie 12: Format d'application musicale interactive

AMENDEMENT 2: Représentation compacte de changement de volume dynamique et égalisation audio

Reference number
ISO/IEC 23000-12:2010/Amd.2:2012(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 23000-12:2010 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 — Multimedia application format (MPEG-A) —

Part 12:
Interactive music application format

AMENDMENT 2: Compact representation of dynamic volume change and audio equalization

In 6.6.5: Preset Box, replace:

```
if(preset_type == 0){
    for(i=0; i<num_preset_elements; i++){
        unsigned int(8) preset_volume_element;

    }

}

if(preset_type == 1){
    unsigned int(8)   num_input_channel[num_preset_elements];
    unsigned int(8)   output_channel_type;
    for (i=0; i<num_preset_elements; i++){
        for (j=0; j<num_input_channel[i]; j++){
            for (k=0; k<num_output_channel; k++){
                unsigened int(8)   preset_volume_element;
            }

        }
    }
}

if(preset_type == 2){ // dynamic track volume preset
    unsigned int(16)   num_updates;
    for(i=0; i<num_updates; i++){
        unsigned int(16)  updated_sample_number;
        for(j=0; j<num_preset_elements; j++){
            unsigned int(8) preset_volume_element;

        }
    }
}

if(preset_type == 3){ // dynamic object volume preset
    unsigned int(16)   num_updates;
    unsigned int(8)   num_input_channel[num_preset_elements];
    unsigned int(8)   output_channel_type;
```

```
        for(i=0; i<num_updates; i++){
            unsigned int(16)  updated_sample_number;
            for(j=0; j<num_preset_elements; j++){
                for(k=0; k<num_input_channel[j]; k++){
                    for(m=0; m<num_output_channel; m++){
                        unsigned int(8)   preset_volume_element;
                    }

                }
            }
        }
    }
```

with:

```
    if(preset_type&0x07 == 0){ // static track volume preset
        for(i=0; i<num_preset_elements; i++){
            unsigned int(8) preset_volume_element;

       if(preset_type&0x08 == 8){ // with EQ
                unsigned int(8)   num_eq_filters;
                for(j=0; j<num_eq_filters; j++){
                    unsigned int(8)  filter_type;
                    unsigned int(16)    filter_reference_frequency;
                    unsigned int(8)     filter_gain;
                    unsigned int(8)  filter_bandwidth;
                }
            }
        }
    }

    if(preset_type&0x07 == 1){ // static object volume preset
        unsigned int(8)   num_input_channel[num_preset_elements];
        unsigned int(8)   output_channel_type;
        for (i=0; i<num_preset_elements; i++){
            for (j=0; j<num_input_channel[i]; j++){
                for (k=0; k<num_output_channel; k++){
                    unsigned int(8)   preset_volume_element;
                }

                if(preset_type&0x08 == 8){ // with EQ
                    unsigned int(8)  num_eq_filters;
                    for(k=0; k<num_eq_filters; k++){
                        unsigned int(8)     filter_type;
                        unsigned int(16)    filter_reference_frequency;
                        unsigned int(8) filter_gain;
                        unsigned int(8)     filter_bandwidth;
                    }
                }
            }
        }
    }

    if(preset_type&0x07 == 2){ // dynamic track volume preset
        unsigned int(16)   num_updates;
        for(i=0; i<num_updates; i++){
            unsigned int(16)  updated_sample_number;
```

```
            for(j=0; j<num_preset_elements; j++){
                unsigned int(8) preset_volume_element;

                if(preset_type&0x08 == 8){ // with EQ
                    unsigned int(8)  num_eq_filters;
                    for(k=0; k<num_eq_filters; k++){
                        unsigned int(8)     filter_type;
                        unsigned int(16)    filter_reference_frequency;
                        unsigned int(8)     filter_gain;
                        unsigned int(8)     filter_bandwidth;
                    }
                }
            }
        }
    }

    if(preset_type&0x07 == 3){ // dynamic object volume preset
        unsigned int(16)    num_updates;
        unsigned int(8)    num_input_channel[num_preset_elements];
        unsigned int(8)    output_channel_type;
        for(i=0; i<num_updates; i++){
            unsigned int(16)  updated_sample_number;
            for(j=0; j<num_preset_elements; j++){
                for(k=0; k<num_input_channel[j]; k++){
                    for(m=0; m<num_output_channel; m++){
                        unsigned int(8)    preset_volume_element;
                    }

                    if(preset_type&0x08 == 8){ // with EQ
                        unsigned int(8)     num_eq_filters;
                        for(m=0; m<num_eq_filters; m++){
                            unsigned int(8)    filter_type;
                            unsigned int(16)   filter_reference_frequency;
                            unsigned int(8)     filter_gain;
                            unsigned int(8)    filter_bandwidth;
                        }
                    }
                }
            }
        }
    }

    if(preset_type&0x07 == 4){ // dynamic track approximated volume preset
        unsigned int(16) num_updates;
        for(i=0; i<num_updates; i++){
            unsigned int(16)  start_sample_number;
            unsigned int(16)  duration_update;
            for(j=0; j<num_preset_elements; j++){
                unsigned int(8) end_preset_volume_element;

                if(preset_type&0x08 == 8){ // with EQ
                    unsigned int(8)  num_eq_filters;
                    for(k=0; k<num_eq_filters; k++){
                   unsigned int(8)   filter_type;
                    unsigned int(16)  filter_reference_frequency;
                          unsigned int(8)   end_filter_gain;
                        unsigned int(8)     filter_bandwidth;
               }
                }
            }
```

```
        }
    }

    if(preset_type&0x07 == 5){ // dynamic object approximated volume preset
        unsigned int(16) num_updates;
        unsigned int(8) num_input_channel[num_preset_elements];
        unsigned int(8) output_channel_type;
        for(i=0; i<num_updates; i++){
            unsigned int(16)  start_sample_number;
            unsigned int(16)  duration_update;
            for(j=0; j<num_preset_elements; j++){
                for(k=0; k<num_input_channel[j]; k++){
                    for(m=0; m<num_output_channel; m++){
                        unsigned int(8) end_preset_volume_element;
                    }

                    if(preset_type&0x08 == 8){ // with EQ
                        unsigned int(8)     num_eq_filters;
                        for(m=0; m<num_eq_filters; m++){
                       unsigned int(8)   filter_type;
                            unsigned int(16)   filter_reference_frequency;
                        unsigned int(8)     end_filter_gain;
                            unsigned int(8)    filter_bandwidth;
                          }
                    }
                }
            }
        }
    }
```

In 6.6.5: Preset Box, replace:

`preset_type` – is an integer that indicates the preset type.

Static track volume preset has the *time invariant* volume information related to each *track* involved in the preset. In this case, the output channel type is the same as channel type of the track which has the largest number of channels among tracks involved in the preset. Type value is 0.

Static object volume preset has the *time invariant* volume information related to each *object* which is individual channel (i.e. mono) of the track involved in the preset. Type value is 1.

Dynamic track volume preset has the *time variant* volume information related to each *track* involved in the preset. In this case, the output channel type is the same as channel type of the track which has the largest number of channels among tracks involved in the preset. Type value is 2.

Dynamic object volume preset has the *time variant* volume information related to each *object* which is individual channel (i.e. mono) of the track involved in the preset. Type value is 3.

preset_type	Meaning
0	static track volume preset
1	static object volume preset
2	dynamic track volume preset
3	dynamic object volume preset

with:

`preset_type` – is an integer that indicates the preset type.

A preset can contain volume and/or audio equalization (EQ) information. The last three bits (0b00000111) of `preset_type` represent the volume related information, and the fourth last bit (0b00001000) represents EQ related information.

Static track volume preset has the *time invariant volume* with or without *EQ information* related to each *track* involved in the preset. In this case, the output channel type is the same as channel type of the track which has the largest number of channels among tracks involved in the preset. Type value is 0 without EQ, or 8 with EQ.

Static object volume preset has the *time invariant volume* with or without *EQ information* related to each *object* which is individual channel (i.e. mono) of the track involved in the preset. Type value is 1 without EQ, or 9 with EQ.

Dynamic track volume preset has the *time variant volume* with or without *EQ information* related to each *track* involved in the preset. In this case, the output channel type is the same as channel type of the track which has the largest number of channels among tracks involved in the preset. Type value is 2 without EQ, or 10 with EQ.

Dynamic object volume preset has the *time variant volume* with or without *EQ information* related to each *object* which is individual channel (i.e. mono) of the track involved in the preset. Type value is 3 without EQ, or 11 with EQ.

Dynamic track approximated volume preset has the *time variant approximated volume* with or without *EQ information* related to each *track* involved in the preset. In this case, the output channel type is the same as channel type of the track which has the largest number of channels among tracks involved in the preset. Type value is 4 without EQ, or 12 with EQ.

Dynamic object approximated volume preset has the *time variant approximated volume* with or without *EQ information* related to each *object* which is individual channel (i.e. mono) of the track involved in the preset. Type value is 5 without EQ, or 13 with EQ.

preset_type	Meaning
0	static track volume preset
1	static object volume preset
2	dynamic track volume preset
3	dynamic object volume preset
4	dynamic track approximated volume preset
5	dynamic object approximated volume preset
6	Value reserved
7	Value reserved
8	static track volume preset with EQ
9	static object volume preset with EQ
10	dynamic track volume preset with EQ
11	dynamic object volume preset with EQ
12	dynamic track approximated volume preset with EQ
13	dynamic object approximated volume preset with EQ

In 6.6.5: Preset Box, replace:

`num_updates` – is an integer that gives the number of updates on `preset_volume`.

with:

`num_updates` – is an integer that gives the number of updates on `preset_volume` or `filter_gain`. In the case of `preset_type == 4, 5, 12 and 13`, it indicates an integer that gives the number of updates on `end_preset_volume` or `end_filter_gain`.

In 6.6.5: Preset Box,previous to "`preset_name` – is a null-terminated string in UTF-8 characters which gives a human-readable name for the preset.", *add:*

`start_sample_number` – is an integer that indicates the time when the gradual volume or EQ update takes place.

`duration_update` – is an integer that indicates the number of samples (time duration) that the gradual volume or EQ update to incur. The volume level or filter gain level is changing linearly in time in the time duration from the previous volume level or filter gain level before the update to the new volume level or filter gain level. When `duration_update` has the value 0, it indicates that the volume or EQ update incurs instantly.

`end_preset_volume_element` – is an integer that indicates the new volume at the end of the gradual volume update.

In dynamic presets (`preset_type` equals to 2, 3, 4, 5, 10, 11, 12 and 13), the first volume update should be at the beginning time (where `updated_sample_number` equals to 0) so that volume levels are defined for the whole time period of the music.

`num_eq_filters` – is an integer representing the number of filters used for each preset element.

`filter_type` – is an integer representing the type of filter. There are 5 filter types: Low pass filter (LPF), High pass filter (HPF), Low shelf filter (LSF), High shelf filter (HSF) and Peaking filter. The filter types and corresponding value of filter_type is listed in the table below.

`filter_type`	1	2	3	4	5
Filter type	LPF	HPF	LSF	HSF	Peaking

`filter_reference_frequency`, `filter_gain`, `end_filter_gain`, and `filter_bandwidth` – are integers representing the parameters of the filters. Exact meanings of the parameters depend on the type of filter, as specified in `filter_type`. The meanings of the parameters are listed in the table below.

Filter type	`filter_reference_frequency`	`filter_gain`/ `end_filter_gain`	`filter_bandwidth`
LPF	Cut-off frequency (F in Hz)	Undefined	Slope (S in dB/octave)
HPF	Cut-off frequency (F in Hz)	Undefined	Slope (S in dB/octave)
LSF	Corner frequency (F in Hz)	Gain (G in dB)	Slope (S in dB/octave)
HSF	Corner frequency (F in Hz)	Gain (G in dB)	Slope (S in dB/octave)
Peaking	Center frequency (F in Hz)	Gain (G in dB)	Quality factor (Q)

`filter_reference_frequency` – is a 16-bit unsigned integer. The frequency value F is exactly the value of filter_reference_frequency: F=filter_reference_frequency (Hz). The frequency range is from 0Hz to 65535Hz, which covers the frequency range of 96kHz sampled audio.

`filter_gain` – is an 8-bit unsigned integer. The gain value G represented by filter_gain is computed by: G=filter_gain/5-41 (dB). A range between -41.0dB to 10.0dB with 0.2dB resolution can be represented. In the cases of low pass filter (LPF) and high pass filter (HPF), filter_gain is undefined. For LPF and HPF, filter gain is undefined.

`end_filter_gain` – is an 8-bit unsigned integer that indicates the EQ filter gain at the end of the gradual EQ update. The gain value G represented by end_filter_gain is computed by: G=filter_gain/5-41 (dB). Thus a range between -41.0dB to 10.0dB with 0.2dB resolution can be represented. In the cases of low pass filter (LPF) and high pass filter (HPF), end_filter_gain is undefined.

`filter_bandwidth` – is an 8-bit unsigned integer, which indicates slope of filter for LPF, HPF, LSF and HSF. The slope value S in dB/octave is computed by: S=filter_bandwidth*6 (dB/octave). filter_bandwidth indicates the quality factor Q for peaking filter. The value Q is computed by: Q=filter_bandwidth/10.

Add the following Annexes after Annex D:

Annex E
(informative)

Compact Dynamic Volume Change Representation

E.1 Description of compact dynamic volume change representation

In dynamic volume preset of IM AF, the volume of a track (or tracks) can vary over time. In the IM AF standard specification, volume change at each sample is represented individually by the sample number and the new volume level. Figure E.1 gives an example of volume fading (for illustration only). S_A and S_B indicate the sample numbers of the starting time and ending time of the fading. V_A and V_B indicate the corresponding volume level of S_A and S_B. The horizontal dashed lines stand for time period with no volume change.

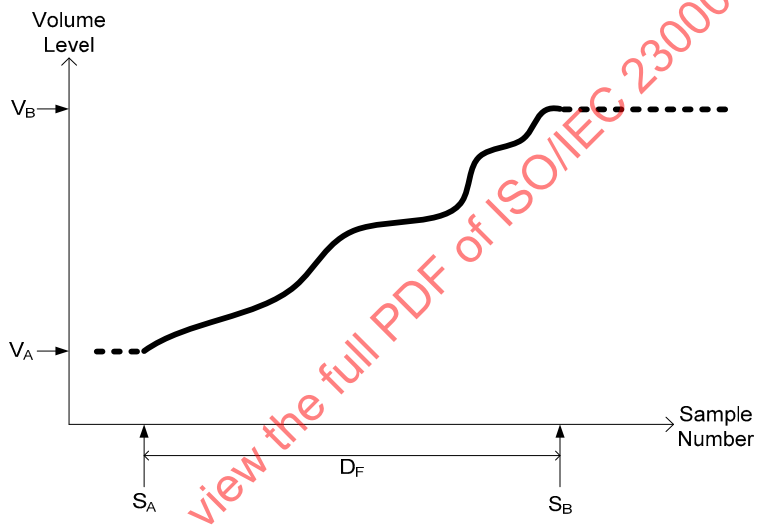

Figure E.1 — Volume curve of fading

By using the existing IM AF standard specification for representation of the dynamic volume changes (preset_type == 2 or 3), D_F pairs of (sample number, new level) are needed, where D_F=S_B-S_A. It should be noted that D_F could be a quite large number, as the time duration of a sample in IM AF is about 21ms for AAC audio with sample rate of 48kHz. A large number D_F would imply a large number of volume updates in the IM AF file.

In the new compact representation, the dynamic volume changes are specified by using time intervals, instead of by using each sample as in the existing representation. The volume change of one time interval is illustrated in Figure E.2. It is assumed that the volume change during the time interval is close to linear. Thus the volume changes during the time interval can be represented by a triplet (*a, b, c*), where *a* stands for the starting sample number, *b* stands for the duration of the time interval (number of samples) that the volume change takes place, and *c* stands for the new volume level at the end of the time interval.

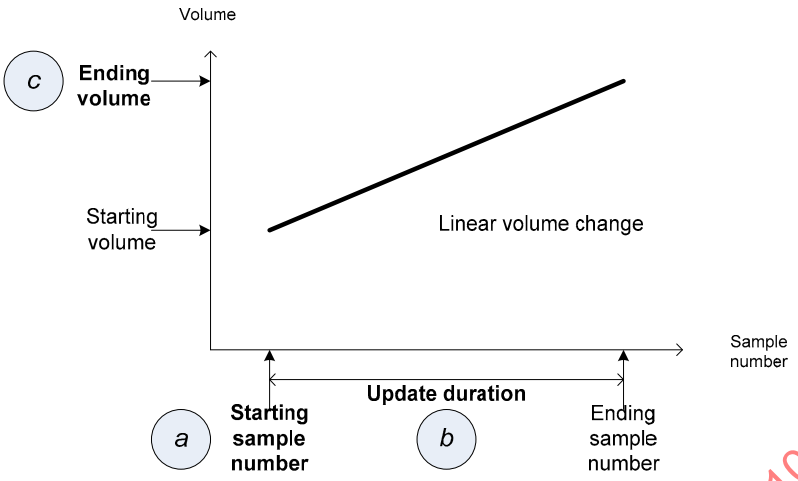

Figure E.2 — Representing volume change by a time interval

For the volume fading example illustrated in Figure E.1, the dynamic volume change information can be represented by about 6 time intervals as shown in Figure E.3. The information required are 6 triplets: (S_1,D_1,V_1), (S_2,D_2,V_2), (S_3,D_3,V_3), (S_4,D_4,V_4), (S_4,D_4,V_4), (S_5,D_5,V_5), and (S_6,D_6,V_6). It should be noted that the volume level V_A is from the previous volume update, and there should always be an instant initial volume update at the beginning time 0: $(0,0,V_0)$.

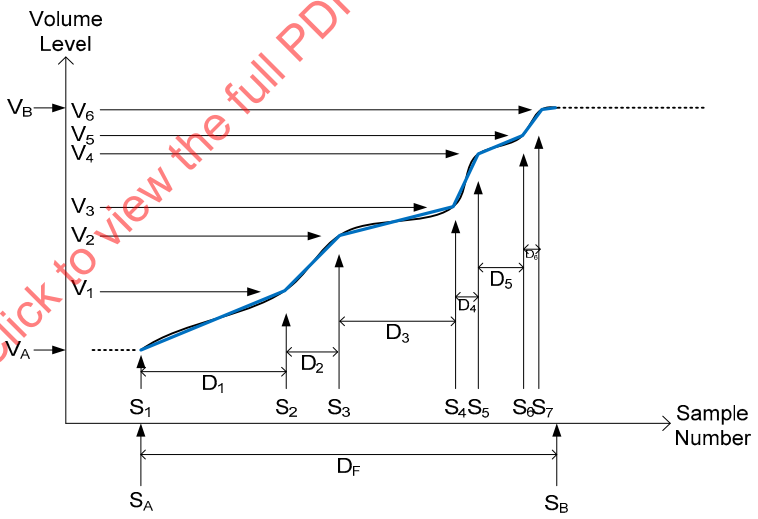

Figure E.3 — Dynamic volume changes represented by the proposed method

With the new compact representation of continuous dynamic volume change, the required storage space in IM AF file and file parsing complexity of IM AF players can be significantly improved, while the errors in the volume levels due to time interval approximation do not affect the audio quality significantly.

E.2 A method for deriving compact representation of dynamic volume change by volume curve approximation

This Subclause presents a method for deriving the compact representation of dynamic volume change in IM AF, which is specified in this amendment, from an arbitrary volume curve. The volume curve may correspond to volume fading, e.g. fade-in and fade-out in various curve shapes, e.g. exponential, logarithmic, linear, etc.

Step 1: Quantizing the volume curve using the original dynamic volume change representation in IM AF.

V[i], where i stands for the index of the samples (assume i takes the values of 1 to N) and V is the volume gain (ratio) that takes the values as indicated in the table for preset_volume_element (0 to 4.00 with step size of 0.02).

Step 2: Approximating V[i] using a sequence of time intervals of linear volume change.

The algorithm below in MATLAB selects the sample indexes T[k] from [i], such that only V[T[k]] are used in the representation of volume change and the unselected samples can be approximated based on the selected samples. The input variable error_thres controls the approximation error.

```
% input: V – volume curve
% input: error_thres
% output: T – indexes of the selected samples

function [T]= fade_coding(V, error_thres)

N=length(V);

point_flag(1:N)=1;

start_i=1;

for i=2:N-1
    if V(start_i)==0
        if V(i)==0
            point_flag(i)=0;
            start_i=i;
            continue;
        else
            start_i=i-1;
            point_flag(i-1)=1;
        end;
    end;

    end_i=i+1;
    flag_error=0;
    for j=start_i+1:end_i-1
        t1=start_i;
        t2=j;
        t3=end_i;
        v1=V(start_i);
        v2=V(j);
        v3=V(end_i);
        v_esti=round(v1+(v3-v1)*(t2-t1)/(t3-t1));
```

```
        v_error=abs(v_esti-v2);
        if v_error > error_thres
            flag_error=1;
            break;
        end;
    end;

    if flag_error==0
        point_flag(i)=0;
    else
        start_i=i;
    end;
end;

T=find(point_flag(1:num_point)==1);
```

The derived compact representation of volume change is a sequence of (T[k], D[k] and Ve[k]), where k stands for the sequence index of time intervals, T stands for the starting sample number of the time interval, D stands for the duration (in number of samples) of the time interval, and Ve stands for the ending volume of the time interval.

D[k]=T[k+1]-T[k]

Ve[k]=V[T[k]]

Annex F
(informative)

Audio Equalization Support in IM AF

F.1 Audio equalization

Audio equalization is a useful feature for users to enhance individual tracks or objects with equalization effects. This functionality is supported by IM AF. An IM AF file can contain a set of tunable equalization parameters for each track or audio object for the desirable effects. The actual equalization processing on each track or object is performed by the filters in the IM AF player.

F.2 An example of efficient implementation of EQ filters in IM AF players

Audio EQ filter can be implemented efficiently using the design based on 2nd order IIR structure.

The transfer function of 2nd order IIR filter is defined as:

$$H(z) = \frac{b_0 + b_1 \times z^{-1} + b_2 \times z^{-2}}{a_0 + a_1 \times z^{-1} + a_2 \times z^{-2}} \quad \text{(F.1)}$$

The flowgraph with b_0 normalized is illustrated in the Figure below.

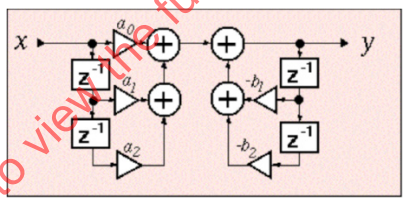

Figure F.1 — Flowgraph of 2nd order IIR filter

The most straightforward implementation of the filters would be the “Direct Form 1” as in Equation F.2 below:

$$y[n] = \left(\frac{b_0}{a_0}\right) \times x[n] + \left(\frac{b_1}{a_0}\right) \times x[n-1] + \left(\frac{b_2}{a_0}\right) \times x[n-2] - \left(\frac{a_1}{a_0}\right) \times y[n-1] - \left(\frac{a_2}{a_0}\right) \times y[n-2] \quad \text{(F.2)}$$

The filter coefficients in Eq. (2) (a_0, a_1, a_2, b_0, b_1 and b_2) can be derived directly from the EQ parameters:

- Reference frequency (F_0),
- Filter gain (G), and
- Filter bandwidth (S or Q)