SPIRIT DSP
Embedded Voice Experience

SPIRIT Parametric Equalizer
User's Guide

Version 1.0

October, 2010

Copyright Information

Read this first

The document describes parametric equalizer software modules and detailed software interface definitions. Document contains examples of the software integration and usage.

To read this document you are not required to have any special knowledge. However, prior experience with C-programming is desirable.

The parametric equalizer software performs equalizing (amplification/attenuation of user-defined frequency bands) of the PCM audio data.

The document includes parametric equalizer algorithm overview, recommendations on the software applications, API description.

Contents

1. Introduction

The purpose of this document is to describe API, integration process and test procedures of parametric equalizer (referred to as equalizer) software.

1.1. Product overview

The equalizer software performs equalizing by serial application of shelving (1^{st} order) and peaking (2^{nd} order) filters to amplify/attenuate 5 user-defined frequency bands. The algorithm operates independently on the "frame by frame" basis. The frame length is a user-defined parameter.

Specification of the equalizer software product is presented in **Table 1**.

Algorithm	Serial connection of shelving (1^{st} order) and peaking (2^{nd} order) filters.
Channels number	1,2
Frame size	Arbitrary
Algorithmic delay	No extra delay, except framing delay.
Band gain range	From –20 to +20 dB
Input and output signal format	Interleaved, linear 16-bit PCM, little-endian format.
Runtime adjustment	Band gain, band width and center frequency for each band

Table 1. Specifications

For more information about other SPIRIT audio processing technologies visit www.spiritDSP.com

SPIRIT reserves the right to make changes to its products to improve the products' technical characteristics without any notice. Customers are advised to obtain the latest version of relevant information to verify that the data is up-to-date before placing the orders.

SPIRIT warrants performance of its products to current specifications in accordance with SPIRIT's standard warranty. Testing and other quality control techniques are utilized to the extent deemed necessary to support this warranty.

1.2. Related products

The software can be efficiently used in conjunction with other products of SPIRIT:

- MPEG-4 AAC Decoder
- MPEG-1 Layer 3 Codec
- Dynamic Range Control
- Sample Rate Converter
- Automatic Gain Control

Since data format is very common, it can be easily interfaced with other Third Party products.

1.3. Document Overview

This document is organized as follows.

Section 0 explains a functional description of the software.

Section 3 describes integration flow, descriptions of structures and interface functions.

2. Functional Description

This section describes equalizer algorithm and provides several recommendations on equalizer usage.

2.1. Algorithm overview

2.1.1. Overall equalizer scheme

The equalizer is a chain of customizable filters. The filter boosts/cuts (amplifies/attenuates) its own frequency band. To boost/cut the low/high edges of the spectrum, the shelving filters are used. To boost/cut the middle bands of the spectrum, the peaking filters are used. Each filter, processes input signal and feeds the output to the next one as depicted on a figure bellow:

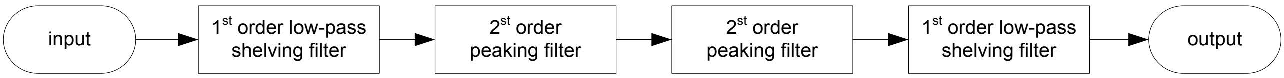

Figure 1. Equalizer block diagram

2.1.2. Shelving filter

The block diagram of the low-pass shelving filter is shown in Figure 2. The *K* parameter defines filter gain: gain is positive if *K* > ½ and is negative if *K* < ½. The shelving filters use the 1st order all-pass filter (AP block in Figure 1), its block diagram is shown in Figure 3.

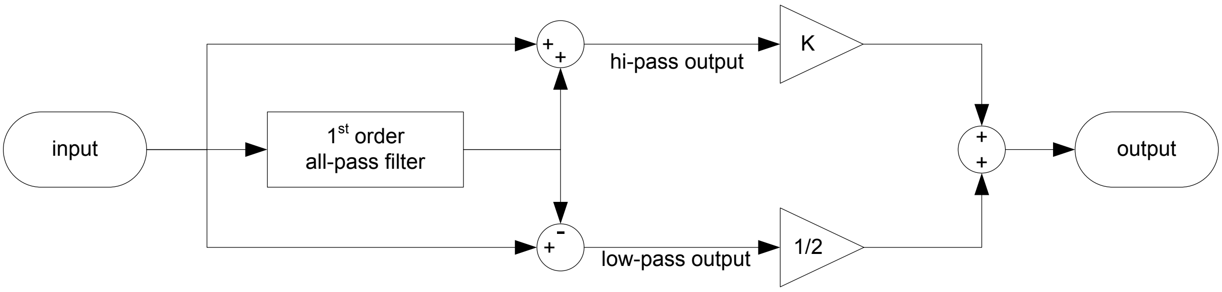

Figure 2. Block diagram of the low-pass shelving filter

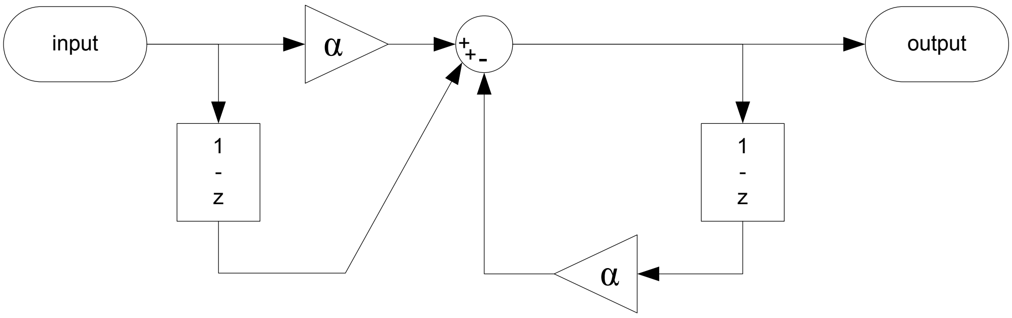

Figure 3. Block diagram of the 1st order all-pass filter used in shelving filters

The transfer function of the 1st order all-pass filter is defined as follows:

$$AP_1(z) = \frac{\alpha + z^{-1}}{1 + \alpha \cdot z^{-1}}$$

The parameter α is used to determine the cut-off frequency and defined as:

$$\alpha = -\frac{1 - \tan\left(\frac{\pi F_0}{F_s}\right)\cdot\gamma}{1 + \tan\left(\frac{\pi F_0}{F_s}\right)\cdot\gamma}$$

where F_0 is the desired cut-off frequency, F_S is the sampling frequency and γ is a scaling factor required for symmetrical response:

$$\gamma = \frac{1}{\sqrt{2\cdot K}}$$

where K is the defined using the desired filter gain in decibels *Gain(dB)*:

$$K = 0.5\cdot 10^{\frac{\text{Gain(dB)}}{20}}$$

The frequency response of low-pass shelving filter for various values of the parameter α is shown in Figure 4.

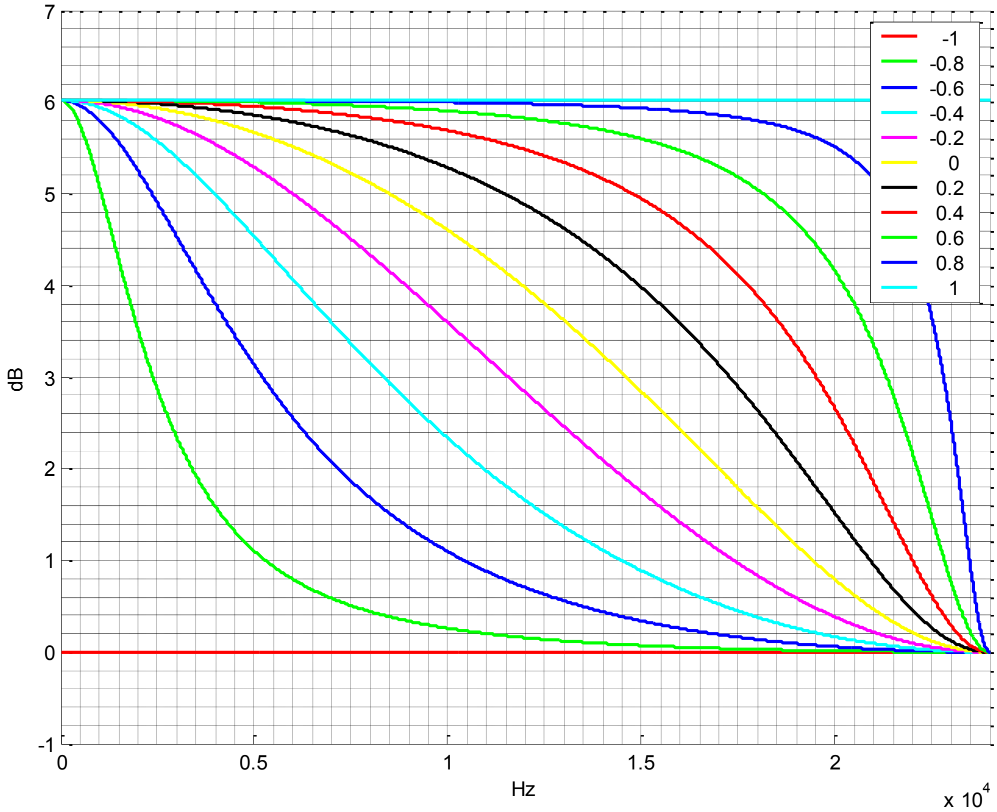

Figure 4. Frequency response of low-pass shelving filter (with +6 dB amplification) for various values of α

2.1.3. Peaking filter

The peaking filter structure is illustrated by the next figures:

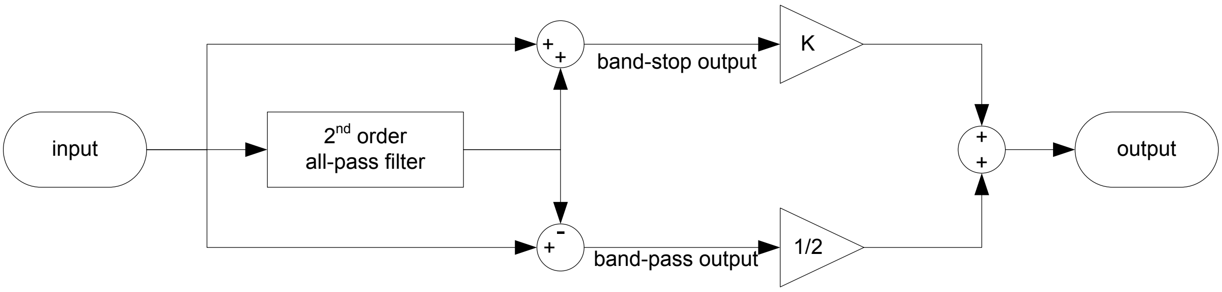

Figure 5. Block diagram of the peaking filter

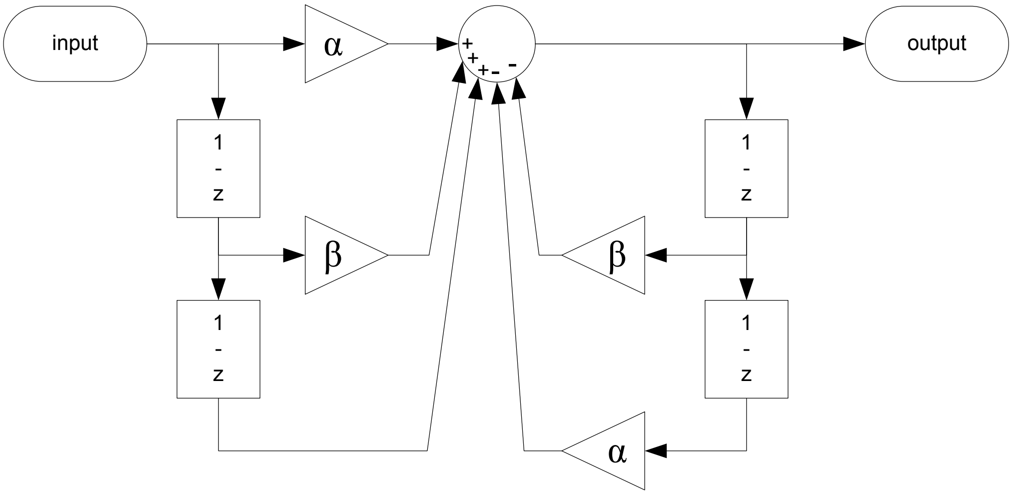

Figure 6. Block diagram of the 2nd order all-pass filter used in peaking filters

The transfer function of the 2nd order all-pass filter is defined as follows:

$$AP_2(z) = \frac{\alpha + \beta \cdot z^{-1} + z^{-2}}{1 + \beta \cdot z^{-1} + \alpha \cdot z^{-2}}$$

The parameter α is used to determine the band width (between –3 dB points around the magnitude response peak) F_0 and defined using exactly the same formulae as for the shelving filter (see Section 2.1.2). The parameter β is used to determine the desired center frequency F_c and is defined as follows:

$$\beta = -\cos\left(\frac{2\pi F_c}{F_s}\right) \cdot (1 + \alpha)$$

The gain parameter K is calculated from the desired filter gain using exactly the same formula as for shelving filter (see Section 2.1.2).

The frequency response of the peaking filter for various values of the parameters α and β is shown in Figure 7 and Figure 8.

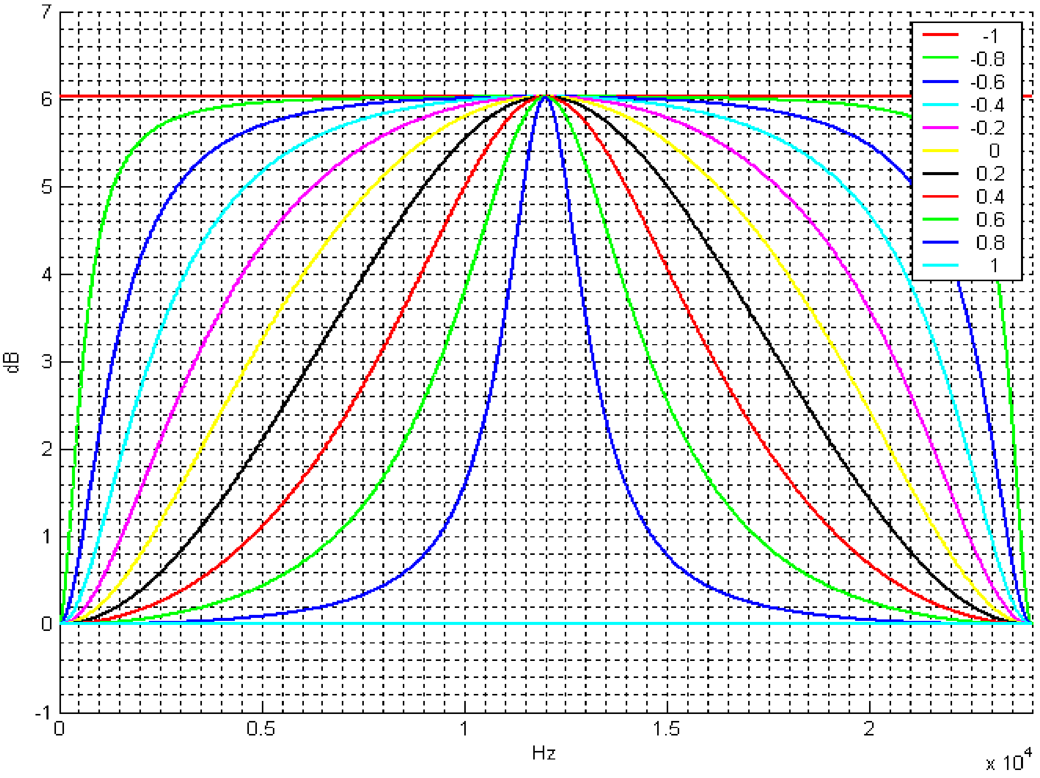

Figure 7. Frequency response of the peaking filter with $\beta = 0$ and various values of α

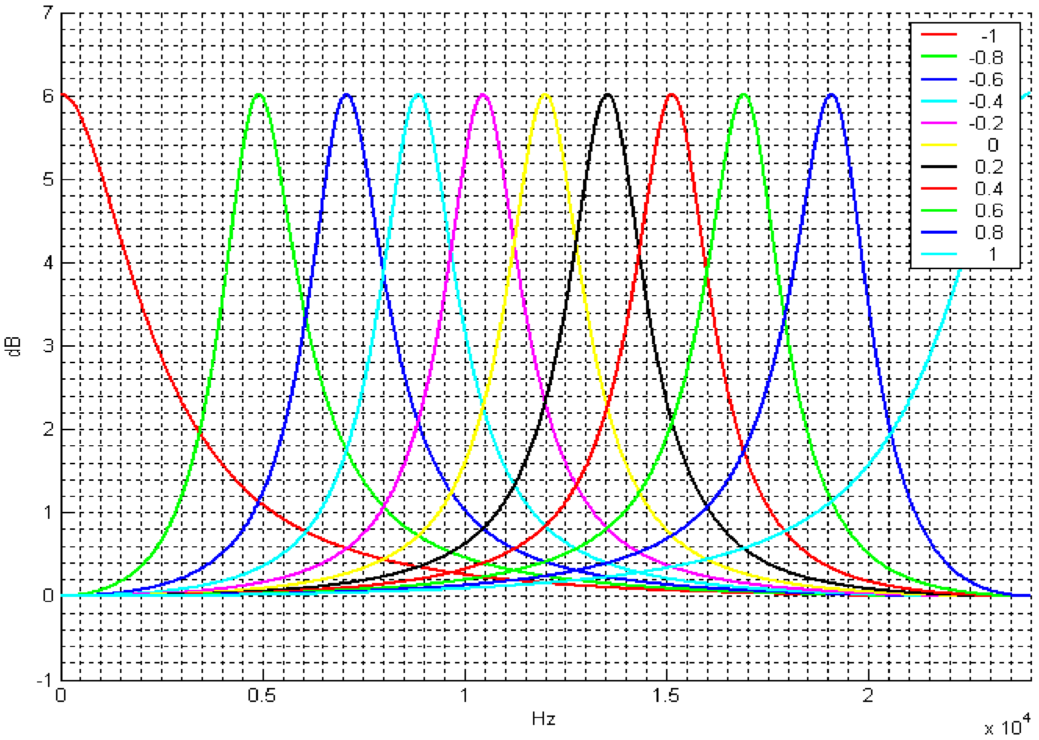

Figure 8. Frequency response of the peaking filter with $\alpha = 0.8$ and various values of β

2.2. Features and recommendations on equalizer usage

2.2.1. Stationary audio enhancement

Figure 9 demonstrates an example of the equalizer software usage where equalizer is used to enhance digital audio data playback/recording by stationary devices (music from AudioCD/DVD (as well as from HDD, flash card, etc), microphone-recorded speech, karaoke devices). The equalizer may be used to suppress high-frequency noise, boost low frequencies to provide more bass, amplifiy/attenuate middle frequency bands to create various audio effects for creative audio recording.

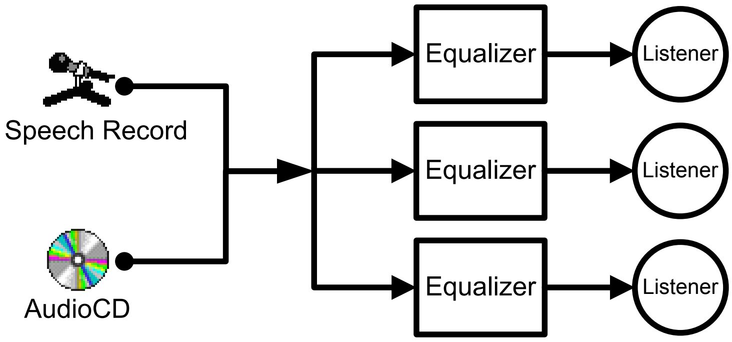

Figure 9. Stationary audio enhancement using equalizer

2.2.2. Mobile audio enhancement

Figure 10 demonstrates example of the equalizer software usage where the audio data (e.g. music tune, speech sample or recorded sound effect attached to MMS message) played back by a mobile device (mobile phone, portable CD/DVD/MP3/WMA/AAC music player) is enhanced by the equalizer. The equalizer enhancement may be used to suppress high-frequency noise, boost low frequencies to provide more bass, amplify/attenuate middle frequency bands to create various creative audio effects, modify audio spectrum to better match frequency response of the headphones and provide more comfortable listening experience.

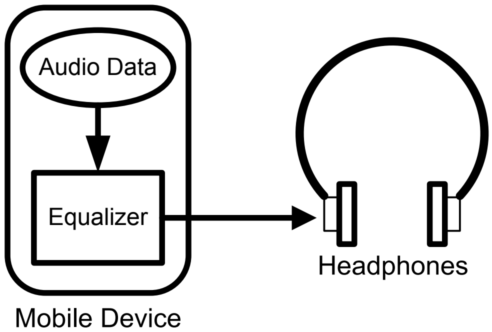

Figure 10. Mobile audio enhancement using equalizer

3. API description

This section describes software implementation and its interface.

3.1. Integration flow

In order to integrate the equalizer into a user's framework, the user should:

Initialize the equalizer using function *SpiritEQ_Init().*

Setup band filters using *SpiritEQ_FltSet().*

Feed each new block of input data to *SpiritEQ_Apply()* function.

3.2. Predefined constants

SPIRIT_EQ_PERSIST_SIZE_IN_BYTES	Persistent memory size in bytes.
SPIRIT_EQ_MAX_BANDS	Maximal number of bands.
SPIRIT_EQ_MAX_CH	Maximal number of bands.
SPIRIT_EQ_MAX_GAIN_DB	Maximal band gain value.
SPIRIT_EQ_MIN_GAIN_DB	Minimal band gain value.
SPIRIT_EQ_FLT_TYPE_NONE	Filter type – disabled.
SPIRIT_EQ_FLT_TYPE_SHELVING_LOWPASS	Filter type – shelving low-pass.
SPIRIT_EQ_FLT_TYPE_SHELVING_HIPASS	Filter type – shelving high-pass.
SPIRIT_EQ_FLT_TYPE_PEAKING	Filter type – peaking.

3.3. TSpiritEQ

Syntax

```
typedef void TSpiritEq;
```

Remarks

Alias of the persistent memory buffer. Only a pointer on this type has sense; all API functions expect it points to a 4 bytes aligned memory region.

3.4. TSpiritEQ_Band

Syntax

```
typedef struct {
    int   fltType;
    int   centerHz;
    int   widthHz;
    short gainDb;
 } TSpiritEQ_Band;
```

Members

fltType	Filter type. Must be one of SPIRIT_EQ_FLT_TYPE_XXX constants.
centerHz	Peaking filter center frequency in Hz.
widthHz	Filter width in Hz.
gainDb	Band in dB (Q0 format) in range [SPIRIT_EQ_MIN_GAIN_DB, SPIRIT_EQ_MAX_GAIN_DB]

Remarks

This structure carries band filter initialization parameters. Please, see section 3.6 for detailed description of the use of this structure.

3.5. SpiritEQ_Init()

Syntax

```
int SpiritEQ_Init     (
          TSpiritEq *eq,
          unsigned long sampleRateHz
);
```

Parameters

eq	Pointer to persistent memory buffer.
sampleRateHz	Sampling rate in Hz.

Return value

Error code.

Remarks

This function initializes equalizer persistent memory buffer as well as disables all band filters and resets filters delay.

3.6. SpiritEQ_FltSet()

Syntax

```
int SpiritEQ_FltSet   (
          TSpiritEq *eq,
          const TSpiritEQ_Band *prms,
          int idx
);
```

Parameters

eq	Pointer to initialized persistent memory buffer.

prms	Band initialization parameters.
idx	Band index in range [0, SPIRIT_EQ_MAX_BANDS-1]

Return value

Error code.

Remarks

This function sets filter coefficients for a chosen band. No error reported if *centerHz*, *widthHz* or *gainDb* exceed their normal range. Instead, max/min values are used. The table bellow summarizes filter parameters conversion based on their values and filter type. Range enclosed in '[]' brackets applied automatically. Fixed value means that parameter is ignored and set to pre-defined value. The '*sf*' term stands for sampling rate.

	None	**Shelving low**	**Shelving high**	**Peaking**
centerHz	0	0	sf/2	[0, sf/2]
widthHz	0	[1, sf/2]	[1, sf/2]	[1, sf/2]
gainDb	0	[max, min]	[max, min]	[max, min]

Note that band parameters can be freely changes in a run-time without audible artifacts only if filter type remains the same or if switching from/to *SPIRIT_EQ_FLT_TYPE_NONE*.

	None	**Shelving low**	**Shelving high**	**Peaking**
None	+	+	+	+
Shelving low	+	+	+	
Shelving high	+	+	+	
Peaking	+			+

For example, switch from low-pass shelving filter to low-band peaking (*centerHz ~= 0*) may cause auditable artifacts but switching from low-pass peaking (*centerHz = 0*) does not.

It is recommended to receive current parameters first and then setup new. The proper call sequence is the following:

```
TSpiritEQ_Band prms;

SpiritEQ_FltGet(obj, &prms);  // get current parameters

prms.xxx = xxx;               // modify
prms.yyy = yyy;

SpiritEQ_SetGet(obj, &prms);  // set new parameters
```

3.7. SpiritEQ_FltGet()

Syntax

```
int SpiritEQ_FltGet    (
            TSpiritEq *eq,
            TSpiritEQ_Band *prms,
            int idx
);
```

Parameters

eq prms idx	Pointer to initialized persistent memory buffer. Band initialization parameters storage. Band index in range [0, SPIRIT_EQ_MAX_BANDS-1]

Return value

Error code.

Remarks

This function returns band filter parameters.

3.8. SpiritEQ_FltReset()

Syntax

```
int SpiritEQ_FltGet    (
          TSpiritEq *eq,
          int idx
);
```

Parameters

eq idx	Pointer to initialized persistent memory buffer. Band index in range [0, SPIRIT_EQ_MAX_BANDS-1]

Return value

Error code.

Remarks

This function reset filter delay for a chosen band.

3.9. SpiritEQ_Apply()

Syntax

```
int SpiritEQ_Apply    (
          TSpiritEq *eq,
          int nChannels,
          short *pcm,
          int nSamplesPerCh
);
```

Parameters

eq	Pointer to initialized persistent memory buffer.

nChannels	Channel number in range [1, SPIRIT_EQ_MAX_CH].
pcm	Input/output PCM buffer.
nSamplesPerCh	Number of input frame samples per channel.

Return value

Error code.

Remarks

This function performs in-place processing of input PCM-buffer. Band filters are applied according to their index value ('0' – goes first, than '1', etc.).

3.10. Error codes

Enumerated name	Description
`SPIRIT_EQ_S_OK`	Operation completed successfully.
`SPIRIT_EQ_E_INVALIDARG`	Bad arguments for a function.
`SPIRIT_EQ_E_SAMPLERATE`	Bad sampling rate
`SPIRIT_EQ_E_MAX_CH`	Bad number of channels.
`SPIRIT_EQ_E_FLT_TYPE`	Bad filter type.
`SPIRIT_EQ_E_BADALIGN_PERSIST`	Bad persistent memory alignment.

3.11. Application example

This is an example of Parametric Equalizer utility usage; it can be linked with mixer library and run.

```
#include <stdio.h>
#include "spiritEQ.h"

static long eq[SPIRIT_EQ_PERSIST_SIZE_IN_BYTES/4];

#define PCM_BUFFER_SIZE_IN_SAMPLES 256
static short bufInOut[PCM_BUFFER_SIZE_IN_SAMPLES];

#define SET_BAND_PRMS(band, _fltType, _centerHz, _widthHz, _gainDb) \
    (band)->fltType  = _fltType;    \
    (band)->centerHz = _centerHz;   \
    (band)->widthHz  = _widthHz;    \
    (band)->gainDb   = _gainDb;

void Process_SingleFile (
    const   char    *szInputName,
    const   char    *szOutputName
)
{
    FILE *pFileIn  = fopen(szInputName ,  "rb");
    FILE *pFileOut = fopen(szOutputName,  "wb");
    unsigned long  lHz = 48000;
    unsigned int   nCh = 2;
    int i;
    TSpiritEQ_Band eq_bands[SPIRIT_EQ_MAX_BANDS] = {0, };

    // Initialize equalizer
    SpiritEQ_Init(eq, lHz);

    // Set band params
    SET_BAND_PRMS(&eq_bands[0], SPIRIT_EQ_FLT_TYPE_SHELVING_LOWPASS ,     0, 1000,   8)
    SET_BAND_PRMS(&eq_bands[1], SPIRIT_EQ_FLT_TYPE_PEAKING          ,  3000, 2000,   5)
    SET_BAND_PRMS(&eq_bands[2], SPIRIT_EQ_FLT_TYPE_PEAKING          , 14000, 6000,   0)
    SET_BAND_PRMS(&eq_bands[3], SPIRIT_EQ_FLT_TYPE_PEAKING          , 20000, 4000,  -6)
    SET_BAND_PRMS(&eq_bands[4], SPIRIT_EQ_FLT_TYPE_SHELVING_HIPASS  , 24000, 4000, -12)

    for(i = 0; i < SPIRIT_EQ_MAX_BANDS; i++) {
        SpiritEQ_FltSet(eq, &eq_bands[i], i);
    }

    while(1) {

        int nSamples = fread(bufInOut, sizeof(short), PCM_BUFFER_SIZE_IN_SAMPLES, pFileIn);
        int nSamplesPerCh = nSamples >> (nCh==2 ?1:0);

        if(nSamplesPerCh == 0) {
            break;
        }

        // Run equalizer
        SpiritEQ_Apply(eq, nCh, bufInOut, nSamplesPerCh);

        // Write output data
        fwrite(bufInOut, sizeof(short), nSamplesPerCh*nCh, pFileOut);
    }

    fclose(pFileIn);
    fclose(pFileOut);
}
```