Fraunhofer Institut Integrierte Schaltungen

ISO/MPEG Layer–3 Encoder Library V3.1 Interface Description

Fraunhofer Institut für Integrierte Schaltungen (Fraunhofer IIS)

Documentation V1.10 8th February 2000

Contents

1 **Scope** 3

2 **Encoder basics** 4
2.1 MPEG audio bit streams 4
2.2 Bitrate switching 4
2.3 Padding Byte 5
2.4 Stereo coding 5
2.5 Sampling frequency and downsampling 6
2.6 Encoder / Decoder delay 6

3 **Interface description** 8

4 **Usage** 9
4.1 Calling sequence 9

5 **Function Reference** 10
5.1 Initialization / De-initialization 10
5.1.1 mp3EncOpen 10
5.1.2 mp3EncClose 11
5.2 Encoder configuration 11
5.2.1 mp3EncGetSupportedFormatList 11
5.2.2 mp3EncSetOutputFormat 12
5.2.3 mp3EncGetCurrentConfiguration 12
5.2.4 mp3EncSetConfiguration 12
5.3 Encoding functions 13
5.3.1 mp3EncEncode 13
5.4 Helper functions 14
5.4.1 mp3EncGetVBRIndexTable 14

6 **Data structures Reference** 15
6.1 mp3EncOutputFormat 15
6.2 mp3EncOutputFormatList 16
6.3 mp3EncConfiguration 17
6.4 Variable bitrate index table 19

A **Technical reference** 21

1 Scope

This document describes the interface and usage of the

ISO/MPEG Layer–3 Encoder Library

developed by the *Fraunhofer Institut für Integrierte Schaltungen* IIS.

2 Encoder basics

To understand all the terms in this document, you are strongly encouraged to read the standard ISO/IEC IS 11172-3 [1] and 13818-3 [2]; this section can only give a rough overview about the ISO/MPEG Layer–3 audio coding standard.

2.1 MPEG audio bit streams

ISO/IEC documents 11172-3 and 13818-3 define the syntax of MPEG 1 and MPEG 2 layer–3 audio bit streams. This encoder implements encoding on the basis of the MPEG 1 / MPEG 2 layer–3 standard.

MPEG 1 / 2 audio coding is based on a time-to-frequency mapping of the signal. The signal is partitioned into overlapping portions and transformed into frequency domain. The spectral components are then quantized and coded.

MPEG 1 defines the coding of waveforms sampled at 48, 44.1 and 32 kHz. Sampling frequencies from 24 kHz through 22.05 kHz down to 16 kHz are covered by the MPEG 2 standard. Sampling rates of 12 kHz, 11.025 kHz and 8 kHz are not standardized; there is, however, a de facto standard set by the FHG IIS called MPEG 2.5. See table 1.

The encoder has to strike a compromise between high signal bandwidth and high compression ratios: It turns out that for the highest compression ratios the lowest sampling rates are needed; i.e. MPEG 2.5 is targeted at very low bit rates down to 8 kBit/s.

Table 1
sampling rates

MPEG 1	MPEG 2	MPEG 2.5
32, 44.1, 48 kHz	16, 22.05, 24 kHz	8, 11.025, 12 kHz
32...320 kBit/s	8...160 kBit/s	8...60 kBit/s

An MPEG layer–3 audio bit stream is composed of frames. Each frame carries information about its size in a 4-bit wide field. Therefore a frame's length can (at a given sampling rate) only take on a set of 16 distinct values.[1]

2.2 Bitrate switching

To generate a bit stream with a non-ISO bitrate, the encoder must intersperse longer (higher-bitrate) and shorter (lower-bitrate) frames to achieve – on the average – the wanted target bitrate. This is called bitrate switching.

1 Actually, only combinations 0001...1110 are allowed, giving a total of 14 values.

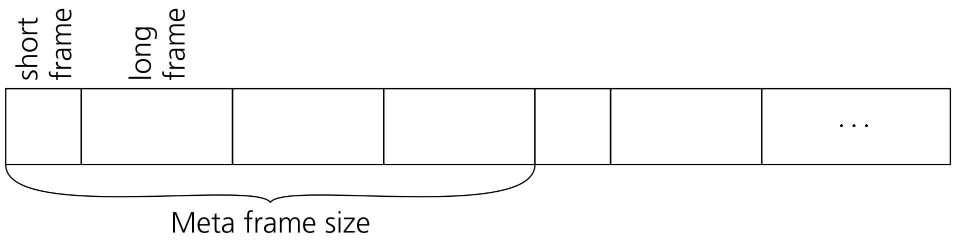

Figure 1
Meta frame sizes and bitrate switching

2.3 Padding Byte

It can easily be seen that each frame of an ISO bitrate bitstream contains

$$n = \frac{samplerate}{bitrate} \cdot \begin{cases} 144 & \text{MPEG 1} \\ 72 & \text{MPEG 2, MPEG 2,5} \end{cases} \tag{1}$$

bytes. ISO bitrates are multiples of 8000 [bit/s]; sampling rates 32000 Hz and 48000 Hz are multiples of 8000 Hz. It is clear therefore that a layer–3 frame contains an integer number of bytes if sample rates of 32 kHz or 48 kHz (or derivatives thereof in the case of MPEG 2 / 2.5) are used.

With sampling rates of 44.1 kHz, 22.05 kHz and 11.025 kHz, (1) is not integer-valued anymore. To achieve an integer byte size for all frames and still hold the nominal ISO bitrate, so-called *padding bytes* are interspersed into the bit stream. The presence of a padding byte is indicated by a bit in the header; a frame with the padding bit set is one byte longer than its bitrate index indicates.

For some applications it may be desirable to be able to chop the bitstream into equally-sized packets that start and stop on frame boundaries. If bitrate switching is used, this cannot be accomplished unless several layer–3 frames are contained in one packet (≫meta frame≪, see figure 1). To get equally-sized meta frames from the encoder, you also need to set the padding to something else than **mp3EncPaddingISO**. (see `mp3EncConfiguration`, section 6.3).

Be aware, however, that these bitstreams do not entirely conform to the ISO/MPEG standard anymore.

2.4 Stereo coding

For coding of binaural waveforms, the standard defines four basic modes:

1. Dual channel. In this mode, both channels are independently coded; no profit is taken of a potentially existing correlation between both channels. This mode is best suited for signals where the channels are independent (e.g. a bilingual source: English left, French right). The encoder library does currently not support use of this mode.

2. Stereo. This mode is similar to the dual channel mode in that still no profit is taken from correlation between channels; the encoder may, however, take bits from a channel that is easy to encode to use them for a hard-to-encode channel.
3. Joint Stereo. In this mode the encoder is allowed to make use of existing correlation between the two channels. It can do this by two means.
   - MS (mid/side) Stereo. Instead of coding the channels independently, the sum and the difference of the left and the right channel are coded. Usually coding of the difference signal needs only a fraction of the bits used on the sum channel so more than half the frame's bits can be spent on the sum channel.
   - Intensity Stereo. While at lower frequencies the spatial perception of the human auditory system relies on magnitude and phase of both stereo signals, at higher frequencies only the energy-time envelopes are evaluated.
     This forms the basis for Intensity Stereo coding: At higher frequencies, only directional and magnitude information is coded for each frequency band.

   If the encoder is in joint stereo mode it can, but does not have to, use MS or Intensity. So although the entire bit stream may be joint stereo, you can encounter any mixture of (plain) stereo, MS-only, IS-only or MS/IS-mixed frames. The encoder's decision is guided by its built-in psychoacoustic module.

2.5 Sampling frequency and downsampling

It has been mentioned above that low bit rates (i.e. high compression ratios) can only be achieved with low sampling rates. If you decide to feed the encoder a high sampling rate while demanding a low bitrate, the encoder needs to downsample the input signal. Because proper downsampling requires filtering the input this will put quite a computational burden on the encoder. You should therefore – if at all possible – change your audio source to a lower sampling rate to relieve the encoder of downsampling.

Since upsampling is not implemented in the encoder, you may not use higher bit rates than the bit rates allowed for your sampling rate's MPEG class, e.g. the highest bit rate for a 22.05 kHz signal is 160 kBit/s.

2.6 Encoder / Decoder delay

The encoder contains a number of delay sources: A preprocessing/downsampling filter, a hybrid filter bank consisting of a polyphase filter and an overlapped MDCT, and a look-ahead buffer within the psychoacoustic module. This delay is implementation dependent and may change from encoder version to encoder version.

If you want all encoder input data to appear in the output bit stream, you need to feed the encoder additional samples (e.g. silence) at the end to compensate the delay.

Since, however, the decoder introduces another delay, you need to append even more samples to the input stream if you want the last sample of your audio data to appear before decoding ends.

Finally, the encoder will refuse to write out incomplete frames, which is why you may need to give it some more samples still.

Fortunately, all this is taken care of by the encoder function `mp3EncEncode()`. If you are out of input data, simply keep calling this function with an input length of zero until it returns no more data. This will ensure that your bit stream contains all the information necessary to decode all of your input data.

3 Interface description

The ISO/MPEG Layer–3 Encoder Library offers a high-level interface for encoding of ISO/MPEG Layer–3 bit streams.

The following header files are provided for usage in C/C++ programs:

mp3encode.h: function prototypes

The encoder core resides in a statically linkable library called libmp3Enc.a (UNIX) or mp3Enc.lib (Microsoft Windows). 32-bit code is used throughout. In `guisample.cpp` you can find example source code demonstrating how to interface with the encoder from a windows application; `demo.c` shows the encoder in a command line application.

4 Usage

4.1 Calling sequence

For encoding of ISO/MPEG Layer–3 bit streams the following sequence is mandatory:

1. Call `mp3EncOpen()` for each encoder instance you need. This call will choose an output format for you that best fits the input format.
2. If you desire to change the output format, call `mp3EncGetSupportedFormatList` to get a list of output formats that the encoder can produce with this input format. Choose one of the supported formats and call `mp3EncSetOutputFormat`. This can be repeated until you are satisfied with the output format.
3. For the expert, each output format offers several options. Some control encoder behaviour (like the audio bandwidth), some bitstream elements (like the SCMS original/copy bit), and some only informational (like the encoder delay). To set the options, call `mp3EncGetCurrentConfiguration`, change the parameters in the structure accessible by the returned pointer, and then call `mp3EncSetConfiguration`.
4. As long as there are still samples left to encode, call `mp3EncEncode()` to encode the data. The encoder returns the bit stream data in a client-supplied buffer.[2]
5. Once you call `mp3EncEncode()` with 0 bytes of input, the flushing process is initiated; afterwards, you may call `mp3EncEncode()` with an input size of zero sample frames only. `mp3EncEncode()` will continue writing out data until all audio samples have been encoded. The bit stream is thus guaranteed to contain all the samples that the encoder received.[3]
6. Once `mp3EncEncode()` has returned with 0 bytes written, call `mp3EncClose()` to destroy this encoder instance. If there are more streams to be generated, go back to 1.

2 It is the client's responsibility to allocate a buffer large enough to hold the output. An 8 KByte buffer will always suffice.

3 Since the encoder has a delay, the decoded bit stream will also contain a number of zero-valued sample frames at the start. The exact delay depends upon the encoder implementation and downsampling factor but is returned by `mp3EncGetCurrentConfiguration()`. See the function reference for details

5 Function Reference

5.1 Initialization / De-initialization

5.1.1 mp3EncOpen

Prototype

```
mp3EncHandle MP3ENCAPI mp3EncOpen
    (
    unsigned int sampleRate,
    unsigned int numChannels,
    mp3EncQualityMode qualityMode
    ) ;
```

Description

Open and initialize one instance of the encoder.

Parameters

- sampleRate
  The samplerate of the encoder input data.
- numChannels
  The number of channels (one or two) of the encoder input data.
- qualityMode
  The encoder quality. Current choices are **mp3EncQualityFast**, **mp3EncQualityMedium**, and **mp3EncQualityHighest**.

Return Value

An encoder handle, initialized and ready-to-go! If anything goes wrong, **NULL** is returned.

Example

→`demo.c`

5.1.2 mp3EncClose

Prototype

```
mp3EncRetval MP3ENCAPI mp3EncClose
    (
    mp3EncHandle hEncoder
    ) ;
```

Description

deallocate an encoder instance

Parameters

– hEncoder
  An encoder handle. It is safe to call this function with a **NULL** handle.

Example

→`demo.c`

5.2 Encoder configuration

5.2.1 mp3EncGetSupportedFormatList

Prototype

```
mp3EncFormatListPtr MP3ENCAPI mp3EncGetSupportedFormatList
    (
    mp3EncHandle hEncoder
    ) ;
```

Description

Get a list of output formats that can be encoded with the input parameters given to `mp3EncOpen()`.

Return Value

A pointer to a format list (see section 6.2) or **NULL** on failure.

5.2.2 mp3EncSetOutputFormat

Prototype

```
mp3EncRetval MP3ENCAPI mp3EncSetOutputFormat
    (
    mp3EncHandle hEncoder,
    mp3EncOutputFormatPtr format
    ) ;
```

Description

Set an output format (samplerate, number of channels, bitrate) for the encoder. You may either fill out the structure yourself or choose one configuration from the list supplied by `mp3EncGetSupportedFormatList()`.

In the former case, you may set the samplerate and/or the number of channels to zero; the encoder will try to find a reasonable setting.

Be aware that this call (as any other call) may fail, indicating invalid output format settings.

5.2.3 mp3EncGetCurrentConfiguration

Prototype

```
mp3EncConfigurationPtr MP3ENCAPI mp3EncGetCurrentConfiguration
    (
    mp3EncHandle hEncoder
    ) ;
```

Description

Get a pointer to a structure describing the current encoder configuration. You may change this structure and feed it into `mp3EncSetConfiguration`.

5.2.4 mp3EncSetConfiguration

Prototype

```
mp3EncRetval MP3ENCAPI mp3EncSetConfiguration
    (
    mp3EncHandle hEncoder,
    mp3EncConfigurationPtr config
    ) ;
```

Description

Set a new configuration. See `mp3EncGetCurrentConfiguration`

5.3 Encoding functions

5.3.1 mp3EncEncode

Prototype

```
    mp3EncRetval MP3ENCAPI mp3EncEncode
        (
        mp3EncHandle hEncoder,
        void *inputBuffer,
        unsigned int bytesInput,
        unsigned int *bytesConsumed,
        void *outputBuffer,
        unsigned int bufferSize
        ) ;
```

Description

encode samples.

Parameters

– hEncoder
  An encoder handle.
– inputBuffer
  Contains audio samples to be encoded. The samples can be of type short (the default) or of type float. When feeding the encoder floats, the values must be in the range $-32768 \ldots 32767$. You need to set the input format to float with a call to `mp3EncSetConfiguration()` in this case.
– bytesInput
  The number of valid bytes in inputBuffer. Once you have called `mp3EncEncode` with 0 bytes of input, the flushing process is initiated. *Do not feed any more data after you have started flushing*.
  When feeding the encoder a stereo signal, the number of audio samples must be an even (i.e., the encoder must be fed complete L/R pairs).
– bytesConsumed
  The number of bytes consumed from the input buffer by the encoder.
– outputBuffer
  Pointer to a buffer receiving the bitstream data. A buffer of 8 KBytes is recommended.
  The buffer does not necessary start of a frame boundary nor does the buffer

necessarily contain a whole number of frames.
– bufferSize
  The size of the output buffer in bytes.

Return Value

A negative number to indicate a failure, the number of valid bytes in the output buffer otherwise. A return value of zero does not indicate failure.

Example

→`demo.c`

5.4 Helper functions

5.4.1 mp3EncGetVBRIndexTable

Prototype

```
mp3EncRetval MP3ENCAPI mp3EncGetVBRIndexTable
    (
    mp3EncHandle hEncoder,
    void *outputBuffer,
    unsigned int bufferSize
    ) ;
```

Description

Retrieve an MPEG Layer–3 frame containing an index table (see 6.4) into the file just encoded. This function can only be used with variable bitrate encoding.

Parameters

– hEncoder
  An encoder handle.
– outputBuffer
  Pointer to a buffer receiving the bitstream data. A buffer of 8 KBytes is recommended.
– bufferSize
  The size of the output buffer in bytes.

Example

→section 6.4 on the format of the VBRI index header.

6 Data structures Reference

6.1 mp3EncOutputFormat

Definition

```
typedef struct mp3EncOutputFormat
{
  unsigned int sampleRate ;
  unsigned int numChannels ;
  unsigned int bitRate ;

} mp3EncOutputFormat, *mp3EncOutputFormatPtr ;
```

Description

Contains information about the encoder output format.

Set an output format (samplerate, number of channels, bitrate) for the encoder. You may either fill out the structure yourself or choose one configuration from the list supplied by `mp3EncGetSupportedFormatList()`.

In the former case, you may set the samplerate and/or the number of channels to zero; the encoder will try to find a reasonable setting.

Fields

– sampleRate
  The samplerate of the encoder output. If you set this to zero, `mp3EncSetOutputFormat` will try to find a reasonable setting.
– bitRate
  The bitrate of the encoder output in $bits/s$. This field **must** be initialized.
– numChannels
  The number of channels in the encoder output. If you set this to zero, `mp3EncSetOutputFormat` will try to find a reasonable setting.

6.2 mp3EncOutputFormatList

Definition

```
typedef struct mp3EncOutputFormatList
{
  unsigned int numEntries ;
  mp3EncOutputFormat format [1] ;

} mp3EncFormatList, *mp3EncFormatListPtr ;
```

Description

Contains a list of suggested encoder output formats. This list does not encompass all the output formats possible; it entails those that are used most often and are recommended.

Fields

– numEntries
  The number of entries in the array `format`.
– format
  An array with `numEntries` entries, containing output format descriptions.

6.3 mp3EncConfiguration

Definition

```
typedef struct mp3EncConfiguration
{
  mp3EncInputFormat inputFormat ;
  mp3EncPaddingMode paddingMode ;

  unsigned int allowIntensity ;
  unsigned int allowMidside ;
  unsigned int allowDownmix ;
  unsigned int writeCRCheck ;

  unsigned int vbrMode ;

  unsigned int privateBit ;
  unsigned int copyrightBit ;
  unsigned int originalBit ;

  float maximumBandwidth ;

  unsigned int codecDelay ;

} mp3EncConfiguration, *mp3EncConfigurationPtr ;
```

Description

Through this structure you can change the encoder configuration.

Fields

– inputFormat
  Tell the encoder whether the input is in IEEE float format (**mp3EncInputFloat**) or in short integers (**mp3EncInputShort**).
– paddingMode
  One of **mp3EncPaddingISO**, **mp3EncPaddingNever**, or **mp3EncPaddingAlways**.
  Certain multimedia architectures (namely, Microsoft Netshow) have a hard time dealing with blocks of variable size. If you want to transport MPEG Layer–3 content over these systems, setting paddingMode to **mp3EncPaddingNever** or **mp3EncPaddingAlways** will make the encoder generate equally-sized metaframes (see section 2.3). If the bitrate b is not one of the ISO bitrates, bitrate switching between frames of bitrate b_0 and frames with bitrate b_1 (where $b_0 \leq b < b1$ and $b_0, b_1 \in$ mpeg bitrates) will be

used. The general rule is that

$$b = \frac{b_0 n_1 + b_1 n_0}{n_0 + n_1}$$

where n_0 (n_1) are the numbers of lower- (higher-) bitrate frames in a meta frame. However, only **mp3EncPaddingISO** is guaranteed to write ISO-compliant bitstreams with exact bitrates.

– allowIntensity
  Usually, you should allow the encoder to use intensity stereo coding, especially at the lower bitrates. However, in special cases, it might be useful to disallow intensity stereo.
– allowMidside
  Usually, you should allow the encoder to use mid/side stereo coding, especially at the lower bitrates. However, in special cases, it might be useful to disallow mid/side stereo. *This flag is not yet supported.*
– allowDownmix
  Usually, you should allow the encoder to downmix stereo input to mono output. However, in special cases, it might be useful to disallow downmix.
– writeCRCheck
  Set this to **1** to have the encoder write CRC checksums.
– vbrMode
  This can be set to a value between 0 and 100. While 0 means *no VBR*, i.e. constant bitrate (the default), values between 1 and 100 produce varying levels of encoder output quality. The highest quality is 100.
  VBR is mutually exclusive with any choice of output format: VBR always produces layer–3 bitstreams with the same sample rate and number of channels as the input.
– privateBit
  This controls the value of the so-called private bit in the MPEG Layer–3 frame header.
– copyrightBit
  Controls the copyright bit in the MPEG frame header. Set this to **1** to indicate that this bitstream is copyright-protected. This implements SCMS (serial copy management system).
– originalBit
  Controls the original/copy bit in the MPEG frame header. Set this to **1** to indicate that this bitstream is an original, to **0** if it is a copy.
– maximumBandwidth
  Tell the encoder to use another bandwidth. Increasing the bandwidth from the default setting will work for some signals, but might produces ringing artefacts for others. Use with consideration! It is not possible to choose bandwidths above half the output sample rate. However, it is recommended to leave the decision to the encoder itself by setting this value to **0**.
– codecDelay
  This read-only parameter field allows an application to retrieve the delay introduced by the encoder. The library will set this value to the number of

Figure 2
VBRI header format

Content	size [bytes]	position
MPEG frame header	4	0
(data indicating silence)	32	4
vbriSignature	4	36
vbriVersion	2	40
vbriDelay	2	42
vbriQuality	2	44
vbriStreamBytes	4	46
vbriStreamFrames	4	50
vbriTableSize	2	54
vbriTableScale	2	56
vbriEntryBytes	2	58
vbriEntryFrames	2	60
vbriTableData[vbriTableSize]	vbriTableSize $\times$ vbriEntryBytes	62

leading zero-valued samples according to the current output format.

6.4 Variable bitrate index table

Variable bitrate bitstreams pose a peculiar problem to the decoder if it supports random access into the file. The problem goes thus: If we want to jump forward, say, 10 seconds, how many bytes in the bitstream does this correspond to? With constant rate bitstreams, this is easy to calculate if you divide the time by the average frame length.

With VBR bitstreams this does not work anymore – the frame length varies too much.

There are two solutions to this problem.

1. If the bitstream is embedded in a real file format, cue points could be provided within the embedding format. This is clearly preferrable, but to our knowledge there are not many players that are capable of playing, say, AIFC-embedded bitstream let alone using the supplied cue points. The *lingua franca* of MPEG audio encoding, as it has turned out, is the bitstream by itself.
2. If the bitstream is used stand-alone, we can provide a workaround. The bitstream is prepended with a pseudo frame that decodes into silence. However, MPEG audio decoders aware of this frame can extract an index table from this frame that serves as a cue point table.

The MPEG Layer–3 audio encoder implements the latter alternative, termed *VBRI* (variable bitrate index) header. Figure 2 shows the layout of the VBRI header.

Fields

- vbriSignature
  4 bytes (»VBRI«)
- vbriVersion
  A version number.
- vbriDelay
  The delay of the sound data relative to the original in units of $\frac{1}{samplerate}$. This delay includes the additional silence caused by the VBRI header itself.
- vbriQuality
  The vbr quality level the file was encoded at.
- vbriStreamBytes
  The total number of bytes in the bitstream. This includes the VBRI header.
- vbriStreamFrames
  The total number of MPEG audio frames in the bitstream. This includes the VBRI header.
  The total duration of the bitstream can be calculated as

  $$\text{duration [sec]} = \frac{vbriStreamFrames \cdot samplesPerFrame}{samplerate}$$

  where

  $$samplesPerFrame = \begin{cases} 1152 & \text{for MPEG-1 bitstreams, i.e. } samplerate >= 32kHz \\ 576 & \text{else} \end{cases}$$
- vbriTableSize
  The number of entries in the table.
- vbriTableScale
  The resolution of the vbriTableData entries. E.g., if resolution==1, the entries are byte-exact, if resolution==4, they are exact to the nearest longword.
- vbriEntryBytes
  The width (in bytes) of each table entry.
- vbriEntryFrames
  The number of frames per table entry.
- vbriTableData
  A table of *vbriTableSize* entries. Each entry represents an equal-duration piece of the bitstream.

To make sure a consistent decoding of the VBRI header is done, we have provided example source code. The source code has been placed in the public domain. An up-to-date version can always be downloaded from `http://www.iis.fhg.de/amm/support/`.

A Technical reference

For the precise definition of most of the terms below, you should consult the ISO/IEC standard. The descriptions below are only accurate to the purposes of this manual.

client The function that calls another function is called its client. If main() calls mp3EncEncode() then main() is mp3EncEncode’s client. Compare with →user.

frame The smallest integral part of an MPEG bit stream. A frame contains two →granules in the case of MPEG 1, one granule else. Consequently, a frame corresponds to 1152 PCM samples in MPEG 1 and to 576 PCM samples else.

granule 576 frequency lines corresponding to 576 PCM samples in the time domain.

sample frame Two →sample points, one left, one right, in the case of stereophonic signals, one in the case of monophonic signals. Successive sample frames are $1/f$ seconds apart where f is the sampling rate in Hz.

sample point One sound sample of one single channel.

user The human that started the program.

References

[1] ISO/IEC JTC1/SC29/WG11 MPEG, International Standard IS 11172-3 Information Technology – Coding of moving pictures and associated audio for digital storage media at up to about 1.5 Mbit/s, Part 3: Audio, 1993.

[2] ISO/IEC JTC1/SC29/WG11 MPEG, International Standard IS 13818-3 Information Technology – Generic Coding of Moving Pictures and Associated Audio, Part 3: Audio, 1994.

[3] M. Bosi, K. Brandenburg, et al. ISO/IEC MPEG-2 advanced audio coding. In *101st AES conference*, Los Angeles, November 1996. preprint 4382.

[4] K. Brandenburg and M. Bosi. Overview of MPEG-audio: Current and future standards for low bit-rate audio coding. In *99th AES conference*, New York, October 1995. preprint 4130.

[5] K. Brandenburg, G. Stoll, et al. The ISO/MPEG-audio codec: A generic standard for coding of high quality digital audio. In Neil Gilchrist and Christer Grewin, editors, *Collected Papers On Digital Audio Bit-Rate Reduction*, pages 31–42. AES, 1996.

[6] R. Buchta, S. Meltzer, et al. The Worldstar sound format. In *101st AES conference*, Los Angeles, November 1996. preprint 4385.

[7] S. Church, B. Grill, H. Popp, et al. ISDN and ISO/MPEG layer–3 audio coding: Powerful new tools for broadcast and audio production. In *95th AES*

conference, Amsterdam, October 1993. preprint 3743.

[8]M. Dietz, H. Popp, et al. Audio compression for network transmission. In *99th AES conference*, New York, October 1995. preprint 4129.

[9]E. Eberlein, H. Popp, et al. Layer–3, a flexible coding standard. In *94th AES conference*, Berlin, March 1993. preprint 3493.

[10]B. Grill, J. Herre, et al. Improved MPEG-2 audio multi-channel encoding. In *96th AES conference*, Amsterdam, February 1994. preprint 3865.

[11]J. Herre, K. Brandenburg, et al. Second generation ISO/MPEG audio layer–3 coding. In *98th AES conference*, Paris, February 1995.

[12]Witte, M. Dietz, et al. Single chip implementation of an ISO/MPEG layer–3 decoder. In *96th AES conference*, Amsterdam, February 1994. preprint 3805.