Ahead AAC Decoder library documentation

Version 1.0.2.1, Menno Bakker (mbakker@nero.com)

1 API specification

This chapter describes the Ahead AAC library API functions and explains how to use them and in which order.

1.1 NeAACDecGetCapabilities

```
unsigned long NEAACAPI NeAACDecGetCapabilities(void);
```

This function returns the capabilities of the decoder in a 32 bit unsigned integer. The bits that are set in the 32 bit unsigned integer define with which capabilities the library has been compiled.
The following capabilities are defined (../include/neaacdec.h)

```
#define LC_DEC_CAP           (1<<0) /* Can decode LC */
#define MAIN_DEC_CAP         (1<<1) /* Can decode MAIN */
#define LTP_DEC_CAP          (1<<2) /* Can decode LTP */
#define LD_DEC_CAP           (1<<3) /* Can decode LD */
#define ERROR_RESILIENCE_CAP (1<<4) /* Can decode ER */
#define FIXED_POINT_CAP      (1<<5) /* Fixed point */
```

This function can be called anytime.

1.2 NeAACDecOpen

```
NeAACDecHandle NEAACAPI NeAACDecOpen(void);
```

Returns a handle to a decoder context.

1.3 NeAACDecClose

```
void NEAACAPI NeAACDecClose(NeAACDecHandle hDecoder);
```

Closes a decoder context that has been opened by NeAACDecOpen.

1.4 NeAACDecGetCurrentConfiguration

```
NeAACDecConfigurationPtr NEAACAPI
NeAACDecGetCurrentConfiguration(NeAACDecHandle hDecoder);
```

Returns the current decoder library configuration.

1.5 NeAACDecSetConfiguration

```
unsigned char NEAACAPI NeAACDecSetConfiguration(NeAACDecHandle
hDecoder, NeAACDecConfigurationPtr config);
```

Sets a new configuration structure for the decoder library.

Return values:
0 – Error, invalid configuration.
1 – OK

1.6 NeAACDecInit

```
long NEAACAPI NeAACDecInit(NeAACDecHandle hDecoder, unsigned char
*buffer, unsigned long buffer_size, unsigned long *samplerate,
unsigned char *channels);
```

Initialises the decoder library using information from the AAC file. The buffer parameter should hold a small part of the AAC file, so that the initialization can be done based on the ADTS or ADIF header. Buffer can also be NULL, but then default initialization parameters will be used.

Return values:
< 0 – Error
>= 0 – Number of bytes read. This amount of bytes should be skipped by the program using the decoder library.
This function fills the samplerate and channels parameters with the detected values.

1.7 NeAACDecInit2

```
char NEAACAPI NeAACDecInit2(NeAACDecHandle hDecoder, unsigned char
*pBuffer, unsigned long SizeOfDecoderSpecificInfo, unsigned long
*samplerate, unsigned char *channels);
```

Initialises the decoder library based on an AudioSpecificConfig as found inside a MP4 file.

Return values:
< 0 – Error
0 - OK
This function fills the samplerate and channels parameters with the detected values.

1.8 NeAACDecDecode

```
void* NEAACAPI NeAACDecDecode(NeAACDecHandle hDecoder,
NeAACDecFrameInfo *hInfo, unsigned char *buffer, unsigned long
buffer_size);
```

Decodes the AAC data passed in buffer.

Returns a pointer to a sample buffer or NULL. Info about the decoded frame is filled in in the NeAACDecFrameInfo structure. This structure holds information about errors during decoding, number of sample, number of channels and samplerate. The returned buffer contains the channel interleaved samples of the frame.

1.9 Structures

1.9.1 NeAACDecConfiguration

```
typedef struct NeAACDecConfiguration
{
    unsigned char defObjectType;
    unsigned long defSampleRate;
    unsigned char outputFormat;
    unsigned char downMatrix;
    unsigned char useOldADTSFormat;
} NeAACDecConfiguration, *NeAACDecConfigurationPtr;
```

Members:

defObjectType: determines the default object type assumed when the library is initialized without any data from the AAC file (eg: when NULL is passed as buffer in NeAACDecInit()). Can be any of the following values:

```
#define MAIN       1 /* MAIN */
#define LC         2 /* Low Complexity (default) */
#define SSR        3 /* Scalable SampleRate */
#define LTP        4 /* Long Term Predition */
#define HE_AAC     5 /* High Efficiency (SBR) */
#define ER_LC     17 /* Error Resilient Low Complexity */
#define ER_LTP    19 /* Error Resilient Long Term Prediction */
#define LD        23 /* Low Delay */
```

defSampleRate: determines the default samplerate assumed when the library is initialized. Default value is 44100.

outputFormat: determines the output format returned by the decoder library. Can be any of the following values:

```
#define FAAD_FMT_16BIT  1 /* 16 bit integers */
#define FAAD_FMT_24BIT  2 /* 24 bit values packed in 32 bit integers
*/
#define FAAD_FMT_32BIT  3 /* 32 bit integers */
#define FAAD_FMT_FLOAT  4 /* single precision floating point */
#define FAAD_FMT_DOUBLE 5 /* double precision floating point */
```

downMatrix: determines whether a 5.1 channel AAC file should be downmatrixed to 2 channel output (value: 1) or whether the output should stay as 5.1 channels (value: 0).

useOldADTSFormat: determines whether the decoder should assume the currently defined 56 bit ADTS header (value: 0) or the 58 bit ADTS header (value: 1) defined in previous versions of the AAC standard. This value should normally always stay at the value 0, it only exists to provide playback capabilities for people that have AAC files with the old header format. All current encoders should output the new ADTS format.

1.9.2 NeAACDecFrameInfo

This structure is returned after decoding a frame and provides info about the decoded frame.

```
typedef struct NeAACDecFrameInfo
{
    unsigned long bytesconsumed;
    unsigned long samples;
    unsigned char channels;
    unsigned char error;
    unsigned long samplerate;
    unsigned char sbr;
    unsigned char object_type;
    unsigned char header_type;
    unsigned char num_front_channels;
    unsigned char num_side_channels;
    unsigned char num_back_channels;
    unsigned char num_lfe_channels;
    unsigned char channel_position[64];
    unsigned char ps;
} NeAACDecFrameInfo;
```

Members:

bytesconsumed: the number of bytes consumed for decoding this frame.

samples: the number of audio samples in this frame. Each channel is counted separately. So when a single channel has 1024 samples and the file has 2 channels, this value will be 2*1024 = 2048.
channels: number of audio channels in this frame
error: contains an error value if an error occurred, 0 otherwise.
samplerate: the samplerate of the frame.
sbr: tells wether sbr is used in this file or not. Can contain any of the following values:

```
#define NO_SBR           0 /* no SBR used in this file */
#define SBR_UPSAMPLED    1 /* upsampled SBR used */
#define SBR_DOWNSAMPLED  2 /* downsampled SBR used */
#define NO_SBR_UPSAMPLED 3 /* no SBR used, but file is upsampled by a
factor 2 anyway */
```

object_type: contains the object type of the AAC file. Can be any of the values as defined in 1.9.1.
header_type: contains the header type of the file that is being decoded. Can contain any of the following values:

```
#define RAW        0 /* No header */
#define ADIF       1 /* single ADIF header at the beginning of the
file */
#define ADTS       2 /* ADTS header at the beginning of each frame */
```

num_front_channels, num_side_channels, num_back_channels, num_lfe_channels: each of these values contain the number of channels of a certain type.
channel_position[64]: contains the position of each of the channels that is returned by the frame decode function. Can contain any of the following values:

```
#define FRONT_CHANNEL_CENTER (1)
#define FRONT_CHANNEL_LEFT   (2)
#define FRONT_CHANNEL_RIGHT  (3)
#define SIDE_CHANNEL_LEFT    (4)
#define SIDE_CHANNEL_RIGHT   (5)
#define BACK_CHANNEL_LEFT    (6)
#define BACK_CHANNEL_RIGHT   (7)
#define BACK_CHANNEL_CENTER  (8)
#define LFE_CHANNEL          (9)
#define UNKNOWN_CHANNEL      (0)
```

ps: PS not used (0) or used (1).

1.10 API usage

The following pseudo-code describes how and in which order to use the different library functions.

```
unsigned long cap = NeAACDecGetCapabilities();
// Check if decoder has the needed capabilities

// Open the library
NeAACDecHandle hAac = NeAACDecOpen();

// Get the current config
NeAACDecConfigurationPtr conf =
NeAACDecGetCurrentConfiguration(hAac);

//
// If needed change some of the values in conf
//

// Set the new configuration
NeAACDecSetConfiguration(hAac, conf);
```

```
// Initialise the library using one of the initialization functions
char err = NeAACDecInit2(hAac, asc, asc_size, &samplerate,
&channels);
if (err != 0)
{
      //
      // Handle error
      //
}

// Loop until decoding finished
do {
      //
      // Put next frame in buffer
      //

      // Decode the frame in buffer
      samplebuffer = NeAACDecDecode(hAac, &hInfo, buffer,
            buffer_size);

      if ((hInfo.error == 0) && (hInfo.samples > 0))
      {
            //
            // do what you need to do with the decoded samples
            //
      } else if (hInfo.error != 0) {
            //
            // Some error occurred while decoding this frame
            //
      }
} while (more data available);

NeAACDecClose(hAac);
```