Buzzy

P. Baillehache

August 2, 2020

Contents

Introduction

Buzzy library is a C library to generate audio output. It is based on the libao library.

It uses the `PBErr` and `GSet` library.

1 Interface

```
// *************** BUZZY.H ***************
#ifndef BUZZY_H
#define BUZZY_H

// Based on the libao library
// https://www.xiph.org/ao/doc/libao-api.html

// ================= Include ==================
```

```
#include <stdlib.h>
#include <stdio.h>
#include <stdbool.h>
#include <math.h>
#include "pberr.h"
#include "gset.h"
#include "ao/ao.h"

// ================= Define ==================

#define BUZZY_B7 3951.0660
#define BUZZY_A7 3520.0000
#define BUZZY_G7 3135.9630
#define BUZZY_F7 2793.8260
#define BUZZY_E7 2637.0200
#define BUZZY_D7 2349.2180
#define BUZZY_C7 2093.0050

#define BUZZY_B6 1975.5330
#define BUZZY_A6 1760.0000
#define BUZZY_G6 1567.9820
#define BUZZY_F6 1396.9130
#define BUZZY_E6 1318.5100
#define BUZZY_D6 1174.6590
#define BUZZY_C6 1046.5020

#define BUZZY_B5 0987.7666
#define BUZZY_A5 0880.0000
#define BUZZY_G5 0783.9909
#define BUZZY_F5 0698.4565
#define BUZZY_E5 0659.2551
#define BUZZY_D5 0587.3295
#define BUZZY_C5 0523.2511

#define BUZZY_B4 0493.8833
#define BUZZY_A4 0440.0000
#define BUZZY_G4 0391.9954
#define BUZZY_F4 0349.2282
#define BUZZY_E4 0329.6276
#define BUZZY_D4 0293.6648
#define BUZZY_C4 0261.6256

#define BUZZY_B3 0246.9417
#define BUZZY_A3 0220.0000
#define BUZZY_G3 0195.9977
#define BUZZY_F3 0174.6141
#define BUZZY_E3 0164.8138
#define BUZZY_D3 0146.8324
#define BUZZY_C3 0130.8128

#define BUZZY_B2 0123.4708
#define BUZZY_A2 0110.0000
#define BUZZY_G2 0097.9988
#define BUZZY_F2 0087.3071
#define BUZZY_E2 0082.4069
#define BUZZY_D2 0073.4162
#define BUZZY_C2 0065.4064

#define BUZZY_B1 0061.7354
#define BUZZY_A1 0055.0000
#define BUZZY_G1 0048.9994
#define BUZZY_F1 0043.6535
```

```
#define BUZZY_E1 0041.2034
#define BUZZY_D1 0036.7081
#define BUZZY_C1 0032.7032

extern const double BUZZY_RANGE[49];

typedef struct Buzzy Buzzy;
typedef struct BuzzyNote BuzzyNote;
#define BUZZY_NOTE_SHAPE void(*noteShape)( \
  Buzzy*, BuzzyNote*, double, double,  double*)

// ================= Data structures ===================

typedef struct Buzzy {

  // Info about the available audio drivers
  ao_info* const* audioDrivers;

  // Number of available audio drivers
  int nbAudioDrivers;

  // ID of the default audio driver
  int idDefaultDriver;

  // ID of the current audio driver
  int idCurDriver;

  // Format of the audio sample
  ao_sample_format sampleFormat;

  // Output audio device
  ao_device* device;

} Buzzy;

typedef struct BuzzyNote {

  // Frequency
  double freq;

  // Amplitude in [0,1]
  double amp;

  // Start time in millisecond
  double start;

  // Duration in millisecond
  double delayMs;

  // Note shape
  BUZZY_NOTE_SHAPE;

} BuzzyNote;

// ================ Functions declaration ====================

// Create a static Buzzy structure
// There must be only one instance of Buzzy at a time
Buzzy BuzzyCreateStatic(void);

// Free the memory used by the static Buzzy 'that' and close the
// attached audio device if necessary
```

```
void BuzzyFreeStatic(Buzzy* that);

// Set the format of the audio sample
// 'nbBits' is the number of bits per sample, must be sizeof(typeof(sample))
// 'rate' is the number of samples per second per channel
// 'nbChannels' is the number of channels (1 for mono, 2 for stereo, ...)
void BuzzySetFormat(
  Buzzy* that,
     int nbBits,
     int rate,
     int nbChannels);

// Print the available audio drivers list on 'stream'
void BuzzyPrintAudioDrivers(
  const Buzzy* that,
        FILE* stream);

// Get the ID of the default audio driver
#if BUILDMODE != 0
static inline
#endif
int BuzzyGetIdDefaultDriver(const Buzzy* that);

// Get the ID of the default audio driver
#if BUILDMODE != 0
static inline
#endif
int BuzzyGetIdDefaultDriver(const Buzzy* that);

// Set the ID of the current audio driver of the Buzzy 'that' to 'id'
// No effect if id is invalid
#if BUILDMODE != 0
static inline
#endif
void BuzzySetIdCurDriver(
  Buzzy* that,
     int id);

// Get the ID of the current audio driver
#if BUILDMODE != 0
static inline
#endif
int BuzzyGetIdCurDriver(const Buzzy* that);

// Get the number of audio drivers
#if BUILDMODE != 0
static inline
#endif
int BuzzyGetNbAudioDriver(const Buzzy* that);

// Get the info about the audio drivers
#if BUILDMODE != 0
static inline
#endif
ao_info* const* BuzzyGetInfoAudioDriver(const Buzzy* that);

// Get the format of the audio sample
#if BUILDMODE != 0
static inline
#endif
ao_sample_format BuzzyGetFormat(const Buzzy* that);
```

```
// Get the number of bits of the audio sample
#if BUILDMODE != 0
static inline
#endif
int BuzzyGetNbBits(const Buzzy* that);

// Get the rate (number of samples per second per channel) of the audio sample
#if BUILDMODE != 0
static inline
#endif
int BuzzyGetRate(const Buzzy* that);

// Get the number of channels of the audio sample
#if BUILDMODE != 0
static inline
#endif
int BuzzyGetNbChannels(const Buzzy* that);

// Open the device of the Buzzy 'that'
void BuzzyOpen(Buzzy* that);

// Close the device of the Buzzy 'that'
void BuzzyClose(Buzzy* that);

// Play a single flat note with frequency 'freq' during 'delayMs' microseconds
// on the Buzzy 'that'
void BuzzyPlaySingleFlatNote(
        Buzzy* that,
  unsigned int delayMs,
         float freq);

// Play a single linear decreasing note with frequency 'freq' during
// 'delayMs' microseconds on the Buzzy 'that'
void BuzzyPlaySingleLinerarDecreasingNote(
        Buzzy* that,
  unsigned int delayMs,
         float freq);

// Calculate the flat value of a sample for the BuzzyNote 'note' at
// time 'timeMs' for the Buzzy 'that' given the max amplitude 'ampMax'
// 'val' must be an array of 'that->nbChannels', the result values are
// stored in 'val'
void BuzzyFlatNote(
      Buzzy* that,
  BuzzyNote* note,
      double ampMax,
      double timeMs,
     double* val);

// Calculate the linearly fading value of a sample for the BuzzyNote 'note' at
// time 'timeMs' for the Buzzy 'that' given the max amplitude 'ampMax'
// 'vals' must be an array of 'that->nbChannels', the result values are
// stored in 'vals'
void BuzzyLinearFadingNote(
      Buzzy* that,
  BuzzyNote* note,
      double ampMax,
      double timeMs,
     double* vals);

// Play the GSet of BuzzyNote 'music' from time 'timeStartMs' to time
// 'timeEndMs' on the Buzzy 'that'
```

```
// The music is automatically scaled ot fit the maximum amplitude
void BuzzyPlayMusic(
  Buzzy* that,
   GSet* music,
   float timeStartMs,
   float timeEndMs);

// Play the whole range from BUZZY_C1 to BUZZY_B7 on the Buzzy 'that'
void BuzzyPlayWholeRange(Buzzy* that);

// Play the Star Wars thema song on the Buzzy 'that'
void BuzzyPlayStarWars(Buzzy* that);

// Return the maximum absolute amplitude of the GSet of BuzzyNote 'music'
// from time 'timeStartMs' to time 'timeEndMs' on the Buzzy 'that'
double BuzzyGetMaxValMusic(
  Buzzy* that,
   GSet* music,
   float timeStartMs,
   float timeEndMs);

// ================ inliner ====================

#if BUILDMODE != 0
#include "buzzy-inline.c"
#endif

#endif
```

2 Code

2.1 buzzy.c

```
// *************** BUZZY.C ***************

// ================= Include ==================
#include "buzzy.h"
#if BUILDMODE == 0
#include "buzzy-inline.c"
#endif

// Global variables
const double BUZZY_RANGE[49] = {

  BUZZY_C1,
  BUZZY_D1,
  BUZZY_E1,
  BUZZY_F1,
  BUZZY_G1,
  BUZZY_A1,
  BUZZY_B1,
  BUZZY_C2,
  BUZZY_D2,
  BUZZY_E2,
  BUZZY_F2,
  BUZZY_G2,
  BUZZY_A2,
  BUZZY_B2,
```

```
  BUZZY_C3,
  BUZZY_D3,
  BUZZY_E3,
  BUZZY_F3,
  BUZZY_G3,
  BUZZY_A3,
  BUZZY_B3,
  BUZZY_C4,
  BUZZY_D4,
  BUZZY_E4,
  BUZZY_F4,
  BUZZY_G4,
  BUZZY_A4,
  BUZZY_B4,
  BUZZY_C5,
  BUZZY_D5,
  BUZZY_E5,
  BUZZY_F5,
  BUZZY_G5,
  BUZZY_A5,
  BUZZY_B5,
  BUZZY_C6,
  BUZZY_D6,
  BUZZY_E6,
  BUZZY_F6,
  BUZZY_G6,
  BUZZY_A6,
  BUZZY_B6,
  BUZZY_C7,
  BUZZY_D7,
  BUZZY_E7,
  BUZZY_F7,
  BUZZY_G7,
  BUZZY_A7,
  BUZZY_B7

};

// ================ Functions implementation ==================

// Create a static Buzzy structure
// There must be only one instance of Buzzy at a time
Buzzy BuzzyCreateStatic(void) {

  // Declare the new Buzzy
  Buzzy that;

  // Initialize the libao
  ao_initialize();

  // Get the list of available audio drivers
  that.audioDrivers = ao_driver_info_list(&(that.nbAudioDrivers));

  // Get the id of the default driver and initialize the current driver
  that.idDefaultDriver = ao_default_driver_id();
  that.idCurDriver = that.idDefaultDriver;

  // Initialise the audio sample format
  memset(
    &(that.sampleFormat),
    0,
    sizeof(ao_sample_format));
```

```
  that.sampleFormat.byte_format =
    (ao_is_big_endian() ? AO_FMT_BIG : AO_FMT_LITTLE);

  // Initialise the device
  that.device = NULL;

  // Return the new Buzzy
  return that;

}

// Free the memory used by the static Buzzy 'that' and close the
// attached audio device if necessary
void BuzzyFreeStatic(Buzzy* that) {

  if (that == NULL) {

    return;

  }

  // Make sur the device is closed
  BuzzyClose(that);

  // Shutdown the libao
  ao_shutdown();

}

// Set the format of the audio sample
// 'nbBits' is the number of bits per sample, must be sizeof(typeof(sample))
// 'rate' is the number of samples per second per channel
// 'nbChannels' is the number of channels (1 for mono, 2 for stereo, ...)
void BuzzySetFormat(
  Buzzy* that,
     int nbBits,
     int rate,
     int nbChannels) {

#if BUILDMODE == 0

  if (that == NULL) {

    BuzzyErr->_type = PBErrTypeNullPointer;
    sprintf(
      BuzzyErr->_msg,
      "'that' is null");
    PBErrCatch(BuzzyErr);

  }

#endif

  // Set the sample's format
  that->sampleFormat.bits = nbBits;
  that->sampleFormat.rate = rate;
  that->sampleFormat.channels = nbChannels;

}

// Print the available audio drivers list on 'stream'
void BuzzyPrintAudioDrivers(
```

```
  const Buzzy* that,
        FILE* stream) {

#if BUILDMODE == 0

  if (that == NULL) {

    BuzzyErr->_type = PBErrTypeNullPointer;
    sprintf(
      BuzzyErr->_msg,
      "'that' is null");
    PBErrCatch(BuzzyErr);

  }

  if (stream == NULL) {

    BuzzyErr->_type = PBErrTypeNullPointer;
    sprintf(
      BuzzyErr->_msg,
      "'stream' is null");
    PBErrCatch(BuzzyErr);

  }

#endif

  // Loop on the audio drivers
  for (
    int i = 0;
    i < BuzzyGetNbAudioDriver(that);
    ++i) {

    fprintf(
      stream,
      "audio driver #%d : %s\n",
      i,
      that->audioDrivers[i]->name);

  }

}

// Open the device of the Buzzy 'that'
void BuzzyOpen(Buzzy* that) {

#if BUILDMODE == 0

  if (that == NULL) {

    BuzzyErr->_type = PBErrTypeNullPointer;
    sprintf(
      BuzzyErr->_msg,
      "'that' is null");
    PBErrCatch(BuzzyErr);

  }

#endif

  // Make sure the device is closed
  BuzzyClose(that);
```

```
  // Open the device with the current format
  that->device =
    ao_open_live(
      that->idCurDriver,
      &(that->sampleFormat),
      NULL);

  if (that->device == NULL) {

    BuzzyErr->_type = PBErrTypeRuntimeError;
    sprintf(
      BuzzyErr->_msg,
      "ao_open_live failed for %s",
      that->audioDrivers[that->idCurDriver]->name);
    BuzzyErr->_fatal = false;
    PBErrCatch(BuzzyErr);
    BuzzyErr->_fatal = true;

  }

}

// Close the device of the Buzzy 'that'
void BuzzyClose(Buzzy* that) {

#if BUILDMODE == 0

  if (that == NULL) {

    BuzzyErr->_type = PBErrTypeNullPointer;
    sprintf(
      BuzzyErr->_msg,
      "'that' is null");
    PBErrCatch(BuzzyErr);

  }

#endif

  if (that->device != NULL) {

    ao_close(that->device);
    that->device = NULL;

  }

}

// Calculate the flat value of a sample for the BuzzyNote 'note' at
// time 'timeMs' for the Buzzy 'that' given the max amplitude 'ampMax'
// 'vals' must be an array of 'that->nbChannels', the result values are
// stored in 'vals'
void BuzzyFlatNote(
      Buzzy* that,
  BuzzyNote* note,
      double ampMax,
      double timeMs,
     double* vals) {

#if BUILDMODE == 0
```

```
  if (that == NULL) {

    BuzzyErr->_type = PBErrTypeNullPointer;
    sprintf(
      BuzzyErr->_msg,
      "'that' is null");
    PBErrCatch(BuzzyErr);

  }

  if (note == NULL) {

    BuzzyErr->_type = PBErrTypeNullPointer;
    sprintf(
      BuzzyErr->_msg,
      "'note' is null");
    PBErrCatch(BuzzyErr);

  }

#endif

  // Get the relative time
  double tNote = timeMs - note->start;

  // Loop on the channels
  for (
    int iChannel = BuzzyGetNbChannels(that);
    iChannel--;) {

    // If the time is inside the note interval
    if (
      tNote >= 0.0 &&
      tNote <= note->delayMs) {

      // Calculate the value
      vals[iChannel] +=
        ampMax * note->amp *
        sin(6.28318 * note->freq * tNote / 1000.0);

    }

  }

}

// Calculate the linearly fading value of a sample for the BuzzyNote 'note' at
// time 'timeMs' for the Buzzy 'that' given the max amplitude 'ampMax'
// 'vals' must be an array of 'that->nbChannels', the result values are
// stored in 'vals'
void BuzzyLinearFadingNote(
      Buzzy* that,
  BuzzyNote* note,
      double ampMax,
      double timeMs,
     double* vals) {

#if BUILDMODE == 0

  if (that == NULL) {

    BuzzyErr->_type = PBErrTypeNullPointer;
```

```
    sprintf(
      BuzzyErr->_msg,
      "'that' is null");
    PBErrCatch(BuzzyErr);

  }

  if (note == NULL) {

    BuzzyErr->_type = PBErrTypeNullPointer;
    sprintf(
      BuzzyErr->_msg,
      "'note' is null");
    PBErrCatch(BuzzyErr);

  }

#endif

  // Get the relative time
  double tNote = timeMs - note->start;

  // Loop on the channels
  for (
    int iChannel = BuzzyGetNbChannels(that);
    iChannel--;) {

    // If the time is inside the note interval
    if (
      tNote >= 0 &&
      tNote <= note->delayMs) {

      // Calculate the fading value
      double fade = 1.0 - tNote / note->delayMs;

      // Calculate the value
      vals[iChannel] +=
        ampMax * note->amp * fade *
        sin(6.28318 * note->freq * tNote / 1000.0);

    }

  }

}

// Play a single flat note with frequency 'freq' during 'delayMs' microseconds
// on the Buzzy 'that'
void BuzzyPlaySingleFlatNote(
        Buzzy* that,
  unsigned int delayMs,
         float freq) {

#if BUILDMODE == 0

  if (that == NULL) {

    BuzzyErr->_type = PBErrTypeNullPointer;
    sprintf(
      BuzzyErr->_msg,
      "'that' is null");
    PBErrCatch(BuzzyErr);
```

```
  }

#endif

  // Create the note
  BuzzyNote note = {

    .amp = 1.0,
    .start = 0.0,
    .delayMs = delayMs,
    .freq = freq,
    .noteShape = BuzzyFlatNote

  };

  // Create the music
  GSet music = GSetCreateStatic();
  GSetAppend(
    &music,
    &note);

  // Play the music
  BuzzyPlayMusic(
    that,
    &music,
    0.0,
    delayMs);

  // Free memory
  GSetFlush(&music);

}

// Play a single linear decreasing note with frequency 'freq' during
// 'delayMs' microseconds on the Buzzy 'that'
void BuzzyPlaySingleLinerarDecreasingNote(
        Buzzy* that,
  unsigned int delayMs,
         float freq) {

#if BUILDMODE == 0

  if (that == NULL) {

    BuzzyErr->_type = PBErrTypeNullPointer;
    sprintf(
      BuzzyErr->_msg,
      "'that' is null");
    PBErrCatch(BuzzyErr);

  }

#endif

  // Create the note
  BuzzyNote note = {

    .amp = 1.0,
    .start = 0.0,
    .delayMs = delayMs,
    .freq = freq,
```

```
    .noteShape = BuzzyLinearFadingNote

  };

  // Create the music
  GSet music = GSetCreateStatic();
  GSetAppend(
    &music,
    &note);

  // Play the music
  BuzzyPlayMusic(
    that,
    &music,
    0.0,
    delayMs);

  // Free memory
  GSetFlush(&music);

}

// Return the maximum absolute amplitude of the GSet of BuzzyNote 'music'
// from time 'timeStartMs' to time 'timeEndMs' on the Buzzy 'that'
double BuzzyGetMaxValMusic(
  Buzzy* that,
   GSet* music,
   float timeStartMs,
   float timeEndMs) {

  // Declare the result amplitude
  double maxVal = 0.0;

  // Get the size of the sample buffer
  double nb =
    (double)BuzzyGetRate(that) *
    (timeEndMs - timeStartMs) / 1000.0;
  size_t nbSamples = (size_t)floor(nb);

  // Calculate the max amplitude of a sample
  double maxAmp = 0.5 *
    pow(
      2.0,
      (double)BuzzyGetNbBits(that));

  // Allocate memory to calculate the value of the samples
  double* vals =
    PBErrMalloc(
      BuzzyErr,
      sizeof(double) * BuzzyGetNbChannels(that));

  // Loop on the sample
  for (
    size_t iSample = 0;
    iSample < nbSamples;
    ++iSample) {

    // Get the time in millisecond of the sample
    double tSample =
      timeStartMs + 1000.0 * ((double)iSample / (double)BuzzyGetRate(that));

    // Reset the sample values
```

```
    for (
      int iChannel = BuzzyGetNbChannels(that);
      iChannel--;) {

      vals[iChannel] = 0.0;

    }

    // Loop on the note of the music
    GSetIterForward iter = GSetIterForwardCreateStatic(music);

    do {

      // Get the note
      BuzzyNote* note = GSetIterGet(&iter);

      // Add the value of the sample for this note
      note->noteShape(
        that,
        note,
        maxAmp,
        tSample,
        vals);

    } while (GSetIterStep(&iter));

    // Loop on the channels
    for (
      int iChannel = BuzzyGetNbChannels(that);
      iChannel--;) {

      if (fabs(vals[iChannel]) > maxVal) {

        maxVal = fabs(vals[iChannel]);

      }

    }

  }

  // Free memory
  free(vals);

  // Return the result amplitude
  return maxVal;

}

// Play the GSet of BuzzyNote 'music' from time 'timeStartMs' to time
// 'timeEndMs' on the Buzzy 'that'
// The music is automatically scaled ot fit the maximum amplitude
void BuzzyPlayMusic(
  Buzzy* that,
   GSet* music,
   float timeStartMs,
   float timeEndMs) {

#if BUILDMODE == 0

  if (that == NULL) {
```

```
    BuzzyErr->_type = PBErrTypeNullPointer;
    sprintf(
      BuzzyErr->_msg,
      "'that' is null");
    PBErrCatch(BuzzyErr);

  }

  if (music == NULL) {

    BuzzyErr->_type = PBErrTypeNullPointer;
    sprintf(
      BuzzyErr->_msg,
      "'music' is null");
    PBErrCatch(BuzzyErr);

  }

#endif

  // Get the size of the sample buffer
  double nb =
    (double)BuzzyGetRate(that) *
    (timeEndMs - timeStartMs) / 1000.0;
  size_t nbSamples = (size_t)floor(nb);
  size_t nbBytes = BuzzyGetNbBits(that) / 8;
  size_t bufferSize =
    nbBytes *
    BuzzyGetNbChannels(that) *
    nbSamples;

  // Allocate memory for the sample buffer
  char* samples =
    PBErrMalloc(
      BuzzyErr,
      bufferSize * nbBytes);

  // Calculate the max amplitude of a sample
  double maxAmp = 0.5 *
    pow(
      2.0,
      (double)BuzzyGetNbBits(that));

  // Allocate memory to calculate the value of the samples
  double* vals =
    PBErrMalloc(
      BuzzyErr,
      sizeof(double) * BuzzyGetNbChannels(that));

  // Get the scale coefficient of the music to avoid overflow
  double maxVal =
    BuzzyGetMaxValMusic(
      that,
      music,
      timeStartMs,
      timeEndMs);
  double scaleAmp = maxAmp / maxVal;

  // Loop on the sample
  for (
    size_t iSample = 0;
    iSample < nbSamples;
```

```
++iSample) {

// Get the time in millisecond of the sample
double tSample =
  timeStartMs + 1000.0 * ((double)iSample / (double)BuzzyGetRate(that));

// Reset the sample values
for (
  int iChannel = BuzzyGetNbChannels(that);
  iChannel--;) {

  vals[iChannel] = 0.0;

}

// Loop on the note of the music
GSetIterForward iter = GSetIterForwardCreateStatic(music);

do {

  // Get the note
  BuzzyNote* note = GSetIterGet(&iter);

  // Add the value of the sample for this note
  note->noteShape(
    that,
    note,
    maxAmp * scaleAmp,
    tSample,
    vals);

} while (GSetIterStep(&iter));

// Loop on the channels
for (
  int iChannel = BuzzyGetNbChannels(that);
  iChannel--;) {

  uint_8 sampleVal8 = (uint_8)floor(vals[iChannel]);
  uint_16 sampleVal16 = (uint_16)floor(vals[iChannel]);
  uint_32 sampleVal32 = (uint_32)floor(vals[iChannel]);

  // Get the offset of the sample in the buffer
  size_t offsetSample =
    iSample * (BuzzyGetNbChannels(that) * nbBytes) +
    iChannel * nbBytes;

  // Loop on the bytes of the sample
  for (
    size_t iByte = nbBytes;
    iByte--;) {

    // Write the byte in the buffer
    switch (nbBytes) {

      case 1:
        samples[offsetSample + iByte] =
          (sampleVal8 >> (iByte * 8)) & 0xff;
        break;

      case 2:
        samples[offsetSample + iByte] =
```

```
            (sampleVal16 >> (iByte * 8)) & 0xff;
          break;

        case 4:
          samples[offsetSample + iByte] =
            (sampleVal32 >> (iByte * 8)) & 0xff;
          break;

        default:
          break;

      }

    }

  }

  // Play the sample
  ao_play(
    that->device,
    samples,
    bufferSize);

  // Free memory
  free(samples);
  free(vals);

}

// Play the whole range from BUZZY_C1 to BUZZY_B7 on the Buzzy 'that'
void BuzzyPlayWholeRange(Buzzy* that) {

#if BUILDMODE == 0

  if (that == NULL) {

    BuzzyErr->_type = PBErrTypeNullPointer;
    sprintf(
      BuzzyErr->_msg,
      "'that' is null");
    PBErrCatch(BuzzyErr);

  }

#endif

  // Create the music
  GSet music = GSetCreateStatic();

  // Loop on the note of the range
  for (
    int iNote = 0;
    iNote < 49;
    ++iNote) {

    // Create the note
    BuzzyNote* note =
      PBErrMalloc(
        BuzzyErr,
        sizeof(BuzzyNote));
```

```
    *note = (BuzzyNote) {

      .amp = 1.0,
      .start = (double)iNote * 500.0,
      .delayMs = 750.0,
      .freq = BUZZY_RANGE[iNote],
      .noteShape = BuzzyLinearFadingNote

    };

    // Add the note to the music
    GSetAppend(
      &music,
      note);

  }

  // Play the music
  BuzzyPlayMusic(
    that,
    &music,
    0.0,
    50000.0);

  // Free memory
  while (GSetNbElem(&music) > 0) {

    BuzzyNote* note = GSetPop(&music);
    free(note);

  }

}

// Play the Star Wars thema song on the Buzzy 'that'
void BuzzyPlayStarWars(Buzzy* that) {

#if BUILDMODE == 0

  if (that == NULL) {

    BuzzyErr->_type = PBErrTypeNullPointer;
    sprintf(
      BuzzyErr->_msg,
      "'that' is null");
    PBErrCatch(BuzzyErr);

  }

#endif

  // Create the music
  GSet music = GSetCreateStatic();

  // Create the notes
  BuzzyNote notes[16];

  notes[0].start = 0.0;
  notes[0].delayMs = 1000.0;
  notes[0].freq = BUZZY_G3;
  notes[1].start = notes[0].start + notes[0].delayMs * 0.75;
  notes[1].delayMs = 1000.0;
```

```
notes[1].freq = BUZZY_D4;
notes[2].start = notes[1].start + notes[1].delayMs * 0.75;
notes[2].delayMs = 166.5;
notes[2].freq = BUZZY_C4;
notes[3].start = notes[2].start + notes[2].delayMs * 0.75;
notes[3].delayMs = 166.5;
notes[3].freq = BUZZY_B3;
notes[4].start = notes[3].start + notes[3].delayMs * 0.75;
notes[4].delayMs = 166.5;
notes[4].freq = BUZZY_A3;
notes[5].start = notes[4].start + notes[4].delayMs * 0.75;
notes[5].delayMs = 1000.0;
notes[5].freq = BUZZY_G4;
notes[6].start = notes[5].start + notes[5].delayMs * 0.75;
notes[6].delayMs = 500.0;
notes[6].freq = BUZZY_D4;
notes[7].start = notes[6].start + notes[6].delayMs * 0.75;
notes[7].delayMs = 166.5;
notes[7].freq = BUZZY_C4;
notes[8].start = notes[7].start + notes[7].delayMs * 0.75;
notes[8].delayMs = 166.5;
notes[8].freq = BUZZY_B3;
notes[9].start = notes[8].start + notes[8].delayMs * 0.75;
notes[9].delayMs = 166.5;
notes[9].freq = BUZZY_A3;
notes[10].start = notes[9].start + notes[9].delayMs * 0.75;
notes[10].delayMs = 1000.0;
notes[10].freq = BUZZY_G4;
notes[11].start = notes[10].start + notes[10].delayMs * 0.75;
notes[11].delayMs = 500.0;
notes[11].freq = BUZZY_D4;
notes[12].start = notes[11].start + notes[11].delayMs * 0.75;
notes[12].delayMs = 166.5;
notes[12].freq = BUZZY_C4;
notes[13].start = notes[12].start + notes[12].delayMs * 0.75;
notes[13].delayMs = 166.5;
notes[13].freq = BUZZY_B3;
notes[14].start = notes[13].start + notes[13].delayMs * 0.75;
notes[14].delayMs = 166.5;
notes[14].freq = BUZZY_C4;
notes[15].start = notes[14].start + notes[14].delayMs * 0.75;
notes[15].delayMs = 1000.0;
notes[15].freq = BUZZY_A3;

for (
  int iNote = 0;
  iNote < 16;
  ++iNote) {

  notes[iNote].amp = 1.0;
  notes[iNote].noteShape = BuzzyLinearFadingNote;
  GSetAppend(
    &music,
    notes + iNote);

}

// Play the music
BuzzyPlayMusic(
  that,
  &music,
  0.0,
```

```
    notes[15].start + notes[15].delayMs);

  // Free memory
  GSetFlush(&music);

}
```

2.2 buzzy-inline.c

```
// *************** BUZZY-INLINE.C ***************

// ================ Functions implementation ====================

// Get the ID of the default audio driver
#if BUILDMODE != 0
static inline
#endif
int BuzzyGetIdDefaultDriver(const Buzzy* that) {

#if BUILDMODE == 0

  if (that == NULL) {

    BuzzyErr->_type = PBErrTypeNullPointer;
    sprintf(
      BuzzyErr->_msg,
      "'that' is null");
    PBErrCatch(BuzzyErr);

  }

#endif

  return that->idDefaultDriver;

}

// Get the ID of the current audio driver
#if BUILDMODE != 0
static inline
#endif
int BuzzyGetIdCurDriver(const Buzzy* that) {

#if BUILDMODE == 0

  if (that == NULL) {

    BuzzyErr->_type = PBErrTypeNullPointer;
    sprintf(
      BuzzyErr->_msg,
      "'that' is null");
    PBErrCatch(BuzzyErr);

  }

#endif

  return that->idCurDriver;

}
```

```
// Set the ID of the current audio driver of the Buzzy 'that' to 'id'
// No effect if id is invalid
#if BUILDMODE != 0
static inline
#endif
void BuzzySetIdCurDriver(
  Buzzy* that,
     int id) {

#if BUILDMODE == 0

  if (that == NULL) {

    BuzzyErr->_type = PBErrTypeNullPointer;
    sprintf(
      BuzzyErr->_msg,
      "'that' is null");
    PBErrCatch(BuzzyErr);

  }

#endif

  if (id >= 0 && id < BuzzyGetNbAudioDriver(that)) {

    that->idCurDriver = id;

  }

}

// Get the number of audio drivers
#if BUILDMODE != 0
static inline
#endif
int BuzzyGetNbAudioDriver(const Buzzy* that) {

#if BUILDMODE == 0

  if (that == NULL) {

    BuzzyErr->_type = PBErrTypeNullPointer;
    sprintf(
      BuzzyErr->_msg,
      "'that' is null");
    PBErrCatch(BuzzyErr);

  }

#endif

  return that->nbAudioDrivers;

}

// Get the info about the audio drivers
#if BUILDMODE != 0
static inline
#endif
ao_info* const* BuzzyGetInfoAudioDriver(const Buzzy* that) {
```

```
#if BUILDMODE == 0

  if (that == NULL) {

    BuzzyErr->_type = PBErrTypeNullPointer;
    sprintf(
      BuzzyErr->_msg,
      "'that' is null");
    PBErrCatch(BuzzyErr);

  }

#endif

  return that->audioDrivers;

}

// Get the format of the audio sample
#if BUILDMODE != 0
static inline
#endif
ao_sample_format BuzzyGetFormat(const Buzzy* that) {

#if BUILDMODE == 0

  if (that == NULL) {

    BuzzyErr->_type = PBErrTypeNullPointer;
    sprintf(
      BuzzyErr->_msg,
      "'that' is null");
    PBErrCatch(BuzzyErr);

  }

#endif

  return that->sampleFormat;

}

// Get the number of bits of the audio sample
#if BUILDMODE != 0
static inline
#endif
int BuzzyGetNbBits(const Buzzy* that) {

#if BUILDMODE == 0

  if (that == NULL) {

    BuzzyErr->_type = PBErrTypeNullPointer;
    sprintf(
      BuzzyErr->_msg,
      "'that' is null");
    PBErrCatch(BuzzyErr);

  }

#endif
```

```
  return that->sampleFormat.bits;

}

// Get the rate (number of samples per second per channel) of the audio sample
#if BUILDMODE != 0
static inline
#endif
int BuzzyGetRate(const Buzzy* that) {

#if BUILDMODE == 0

  if (that == NULL) {

    BuzzyErr->_type = PBErrTypeNullPointer;
    sprintf(
      BuzzyErr->_msg,
      "'that' is null");
    PBErrCatch(BuzzyErr);

  }

#endif

  return that->sampleFormat.rate;

}

// Get the number of channels of the audio sample
#if BUILDMODE != 0
static inline
#endif
int BuzzyGetNbChannels(const Buzzy* that) {

#if BUILDMODE == 0

  if (that == NULL) {

    BuzzyErr->_type = PBErrTypeNullPointer;
    sprintf(
      BuzzyErr->_msg,
      "'that' is null");
    PBErrCatch(BuzzyErr);

  }

#endif

  return that->sampleFormat.channels;

}
```

3 Makefile

```
# Build mode
# 0: development (max safety, no optimisation)
# 1: release (min safety, optimisation)
# 2: fast and furious (no safety, optimisation)
BUILD_MODE?=0
```

```
all: pbmake_wget main

# Automatic installation of the repository PBMake in the parent folder
pbmake_wget:
if [ ! -d ../PBMake ]; then wget https://github.com/BayashiPascal/PBMake/archive/master.zip; unzip master.zip; rm -f

# Check code style
style:
cbo *.h *.c

# Makefile definitions
MAKEFILE_INC=../PBMake/Makefile.inc
include $(MAKEFILE_INC)

# Rules to make the executable
repo=buzzy
$($(repo)_EXENAME): \
$($(repo)_EXENAME).o \
$($(repo)_EXE_DEP) \
$($(repo)_DEP)
$(COMPILER) `echo "$($(repo)_EXE_DEP) $($(repo)_EXENAME).o" | tr ' ' '\n' | sort -u` $(LINK_ARG) $($(repo)_LINK_ARG)

$($(repo)_EXENAME).o: \
$($(repo)_DIR)/$($(repo)_EXENAME).c \
$($(repo)_INC_H_EXE) \
$($(repo)_EXE_DEP)
$(COMPILER) $(BUILD_ARG) $($(repo)_BUILD_ARG) `echo "$($(repo)_INC_DIR)" | tr ' ' '\n' | sort -u` -c $($(repo)_DIR)/$

install_libao:
cd ~/ ; wget http://downloads.xiph.org/releases/ao/libao-1.2.0.tar.gz ; tar xvf libao-1.2.0.tar.gz ; cd libao-1.2.0

install_sox:
sudo apt install sox

wav2raw:
echo "sox test.wav --bits 16 --rate 44100 --channels 2  test.raw"

reset_device:
pulseaudio -k && sudo alsa force-reload

edit_libao_conf:
sudo gedit /etc/libao.conf
```

4 Unit tests

```
#include <stdlib.h>
#include <stdio.h>
#include <time.h>
#include <string.h>
#include <time.h>
#include <unistd.h>
#include <sys/time.h>
#include "pberr.h"
#include "buzzy.h"

void UnitTestBuzzyCreateFree() {
```

```
  Buzzy buzz = BuzzyCreateStatic();
  if (
    buzz.audioDrivers == NULL ||
    buzz.idDefaultDriver != buzz.idCurDriver ||
    buzz.device != NULL) {

    BuzzyErr->_type = PBErrTypeUnitTestFailed;
    sprintf(
      BuzzyErr->_msg,
      "BuzzyCreateStatic failed");
    PBErrCatch(BuzzyErr);

  }

  BuzzyFreeStatic(&buzz);

  printf("UnitTestBuzzyCreateFree OK\n");

}

void UnitTestBuzzyGetSet() {

  Buzzy buzz = BuzzyCreateStatic();
  ao_sample_format format = BuzzyGetFormat(&buzz);
  int retcmp =
    memcmp(
      &format,
      &(buzz.sampleFormat),
      sizeof(ao_sample_format));
  if (retcmp != 0) {

    BuzzyErr->_type = PBErrTypeUnitTestFailed;
    sprintf(
      BuzzyErr->_msg,
      "BuzzyGetFormat failed");
    PBErrCatch(BuzzyErr);

  }

  if (BuzzyGetIdCurDriver(&buzz) != buzz.idCurDriver) {

    BuzzyErr->_type = PBErrTypeUnitTestFailed;
    sprintf(
      BuzzyErr->_msg,
      "BuzzyGetIdCurDriver failed");
    PBErrCatch(BuzzyErr);

  }

  if (BuzzyGetIdDefaultDriver(&buzz) != buzz.idDefaultDriver) {

    BuzzyErr->_type = PBErrTypeUnitTestFailed;
    sprintf(
      BuzzyErr->_msg,
      "BuzzyGetIdDefaultDriver failed");
    PBErrCatch(BuzzyErr);

  }

  if (BuzzyGetInfoAudioDriver(&buzz) != buzz.audioDrivers) {

    BuzzyErr->_type = PBErrTypeUnitTestFailed;
```

```
  sprintf(
    BuzzyErr->_msg,
    "BuzzyGetInfoAudioDriver failed");
  PBErrCatch(BuzzyErr);

}

if (BuzzyGetNbAudioDriver(&buzz) != buzz.nbAudioDrivers) {

  BuzzyErr->_type = PBErrTypeUnitTestFailed;
  sprintf(
    BuzzyErr->_msg,
    "BuzzyGetNbAudioDriver failed");
  PBErrCatch(BuzzyErr);

}

BuzzySetFormat(
  &buzz,
  16,
  44100,
  2);
if (
  buzz.sampleFormat.bits != 16 ||
  buzz.sampleFormat.rate != 44100 ||
  buzz.sampleFormat.channels != 2) {

  BuzzyErr->_type = PBErrTypeUnitTestFailed;
  sprintf(
    BuzzyErr->_msg,
    "BuzzySetFormat failed");
  PBErrCatch(BuzzyErr);

}

if (BuzzyGetNbBits(&buzz) != buzz.sampleFormat.bits) {

  BuzzyErr->_type = PBErrTypeUnitTestFailed;
  sprintf(
    BuzzyErr->_msg,
    "BuzzyGetNbBits failed");
  PBErrCatch(BuzzyErr);

}

if (BuzzyGetRate(&buzz) != buzz.sampleFormat.rate) {

  BuzzyErr->_type = PBErrTypeUnitTestFailed;
  sprintf(
    BuzzyErr->_msg,
    "BuzzyGetRate failed");
  PBErrCatch(BuzzyErr);

}

if (BuzzyGetNbChannels(&buzz) != buzz.sampleFormat.channels) {

  BuzzyErr->_type = PBErrTypeUnitTestFailed;
  sprintf(
    BuzzyErr->_msg,
    "BuzzyGetNbChannels failed");
  PBErrCatch(BuzzyErr);
```

```
  }

  BuzzySetIdCurDriver(
    &buzz,
    1);
  if (BuzzyGetIdCurDriver(&buzz) != 1) {

    BuzzyErr->_type = PBErrTypeUnitTestFailed;
    sprintf(
      BuzzyErr->_msg,
      "BuzzySetIdCurDriver failed");
    PBErrCatch(BuzzyErr);

  }

  BuzzyFreeStatic(&buzz);

  printf("UnitTestBuzzyGetSet OK\n");

}

void UnitTestBuzzyPlayNote() {

  Buzzy buzz = BuzzyCreateStatic();
  BuzzySetFormat(
    &buzz,
    16,
    44100,
    2);
  BuzzyOpen(&buzz);
  printf("BuzzyPlaySingleFlatNote\n");
  BuzzyPlaySingleFlatNote(
    &buzz,
    1000,
    BUZZY_A4);
  printf("BuzzyPlaySingleLinerarDecreasingNote\n");
  BuzzyPlaySingleLinerarDecreasingNote(
    &buzz,
    1000,
    BUZZY_A3);
  printf("BuzzyPlayWholeRange\n");
  BuzzyPlayWholeRange(&buzz);
  printf("BuzzyPlayStarWars\n");
  BuzzyPlayStarWars(&buzz);
  BuzzyClose(&buzz);
  BuzzyFreeStatic(&buzz);
  printf("UnitTestBuzzyPlayNote OK\n");

}

void UnitTestAll() {

  UnitTestBuzzyCreateFree();
  UnitTestBuzzyGetSet();
  UnitTestBuzzyPlayNote();
  printf("UnitTestAll OK\n");

}

int main() {
```

```
  UnitTestAll();

  // Return success code
  return 0;

}
```

5 Unit tests output

```
UnitTestBuzzyCreateFree OK
UnitTestBuzzyGetSet OK
BuzzyPlaySingleFlatNote
BuzzyPlaySingleLinerarDecreasingNote
BuzzyPlayWholeRange
BuzzyPlayStarWars
UnitTestBuzzyPlayNote OK
UnitTestAll OK
```