UNIVERSITY OF BURGUNDY

Qt Based Sound Recorder

C++ Project Report

Diego NUNES DE ALMEIDA
Yannick PORTO

Bachelor in Computer Vision

May 15, 2015

Abstract

This report presents our work as part of the Computer Science course module, in the second semester of study in the Bachelor in Computer Vision at the University of Burgundy, France. The course objective is to present the principles of object-oriented programming and to introduce Qt as a powerful development framework for application and UI. Our proposal consisted in to develop a Sound Recording application in Qt, which should be able to open and play a pre-existing wave file from the hard drive, record from the audio input (microphone jack or another device) and perform some basic editing tasks. These capabilities, their implementation, and the research necessary for the development of the project are presented.

Contents

1 Introduction **3**

2 Digital Audio Principles **4**

2.1 Terms 4

2.2 Format Summary 6

2.3 The WAVE file 6

3 Qt Audio Classes Overview **8**

4 Functionalities **10**

4.1 The GUI 10

4.2 Menus and Toolbars 11

4.3 The Settings Dialog Box 12

4.4 Recording Track, Backing Track and Output group boxes 12

4.5 The Waveform Scroll Area 13

5 Implementation **14**

5.1 Project Overview 14

5.2 Classes overview 15

6 Drawing the Waveform **17**

7 SoundTrack Class **19**

7.1 Functions, slots and signals 19

7.2 Object members 20

8 SoundEngine Class **21**

8.1 Functions, slots and signals 21

8.2 Object members 22

9 Conclusion **23**

References **24**

A Trello's board **25**

List of Figures

4.1 GUI while recording 10
4.2 Menus and Toolbars of SoundRecorder 11
4.3 Settings Dialog 12
4.4 Tracks and Output group boxes 12
4.5 Waveform scroll area 13

5.1 Project directory 14
5.2 Classes relations 15

6.1 Waveform displaying scheme 17

Chapter 1

Introduction

The objective of this project assignment was to promote an intensive interaction with Qt environment and to practice the concepts of object-oriented programming. Our proposal was motivated by the fact that we are weekend musicians and there is a stage in the musical instrument learning process where is exciting to have the possibility to hear something recorded by yourself.

The proposal consisted in a Sound Recorder application, with the following capabilities:

- Recognition of the audio devices in the computer.
- Sound track for recording audio, with monitor, volume (for playing) and mute (for playing).
- Sound track for playing a backing track, imported from a wave file, with volume and mute.
- Widgets for displaying the intensity level of the sound in both soundtracks and in the output.
- Widget for displaying the waveform of both sound tracks, with a common time line marker and with mouse position extraction.
- Basic editing tasks, namely: global volume, balance control, cutting, fad-in and fad-out.
- Save the edited audio data as a playable wave file.
- Well designed graphic user interface.

Note: By the time of the submission of this report, the audio editing and exporting tasks were not implemented. The implementation will be carried out and presented in the demonstration.

Chapter 2

Digital Audio Principles

Today, sound processing is quite different from what it was a few decades ago. A large portion of sound processing is now done by digital devices and software. As it is captured, processed, and played back, an audio signal can be transformed numerous times from analog-to-digital or digital-to-analog. A basic scenario is given by a microphone – an analog device – continuously detecting changes in the air pressure, converting sound to an analog electrical signal, then an analog-to-digital converter (ADC) converts the analog signal into a digital signal. A digital-to-analog converter performs the reverse process, converting a digital signal back into an analog signal, which analog circuits amplify and send to a loudspeaker.

2.1 Terms

Sampling

Sampling is the act of measuring sound amplitude at equally-spaced moments in time, where each measurement constitutes a sample. The number of samples taken per second (samples/s) is the sampling rate.

The quality of reproducing an analogue signal is increased with a higher sampling rate, as more information is captured. The maximum frequency audible to the human ear is approximately 20kHz. Therefore, in keeping with the Nyquist-Shannon theorem, the sampling rate needed to capture all of the audio in this range needs to be approximately 40KHz. Due to the practicalities of building these electrical systems, the most commonly used baseline sample rate is 44.1kHz. Higher sample rates are being used more within pro-audio systems. 48kHz, 96kHz, and 192kHz are becoming more commonly used as the CD is starting to be replaced by media capable of supporting high sample rates.

This project works only with 44.1kHz as sample rate. Qt's audio interfaces allow a broader exploration of the audio hardware, but it would unnecessarily increase the complexity of the project.

Quantization

Quantization is the way of representing the amplitude of individual samples as integers expressed in binary. The bit depth (or sample size) determines the accuracy of the amplitude measurements per sample. A higher bit depth allows greater audible perception in the the dyanmic range. This is a measurement of the ratio of the loudest available undistorted noise and the quietest available.

16-bit sampling is the standard baseline for consumer level audio (used in CD audio) and was the sample size used in this project.

Sample Type

Sample type indicates whether the samples are signed integers or unsigned integers. As standard, 16-bit encoded audio uses signed integers for representing the samples.

Byte Order

Byte order, also described by the terms endian and endianness, refers to the convention used to interpret the bytes making up a data word when those bytes are stored in computer memory. Each byte of data in the memory has its own address. Big-endian systems store the most significant byte of a word in the smallest address and the least significant byte is stored in the largest address (most significant bit). Little-endian systems, in contrast, store the least significant byte in the smallest address.

After some research was noticed that, for audio files, the most common byte order in use is Little-endian. Thus, this one was used in this project.

Number of Channels

The number of channels indicate how many separate recording elements are encoded in an audio file. A one-channel audio is commonly called mono, and a two-channel is called stereo. In this project, the Recorder Track will allow only recording in mono. In the Backing Track it is possible to import audio with one or two channels.

Bit Rate

The bit rate is the expression of the amount of information that is stored/processed per second. A simple equation can be used to calculate the bit rate of a file when the following values are known:

$$bitRate = bitDepth \times samplingRate \times channels \tag{2.1}$$

Therefore, the byte rate is given by:

$$byteRate = \frac{bitRate}{8} \tag{2.2}$$

PCM

PCM (Pulse Code Modulation) is the basic format for uncompressed audio. A linear PCM scheme stores binary integer values for each sample.

2.2 Format Summary

Below are the audio specifications in use:

- Sample rate = *44100 Hz*
- Bit depth = *16-bits*
- Sample type = *Signed integer*
- Number of channels = *1 or 2*
- Byte rate = *88200 or 176400 bytes/s*
- Byte order = *Little-Endian*
- Format = *PCM*

2.3 The WAVE file

WAVE is a Microsoft and IBM audio file format standard for storing audio bitstreams on computers. It is an application of the Resource Interchange File Format (RIFF) bitstream format method for storing data in “chunks”. A RIFF file is a tagged file format. It has a specific container format (a chunk) that includes a four character tag and the size (number of bytes) of the chunk. Thus, a WAVE file contains a header and the raw data. The header, the beginning of a WAVE (RIFF) file, is used to provide specifications on the file type, sample rate, sample size and bit size of the file, as well as its overall length. The header of a WAV (RIFF) file is 44 bytes long and has the following format:

Interval	**Value**	**Description**
1-4	“RIFF”	Marks the file as a riff file. Characters are each 1 byte long.
5-8	File size	File type - 8 bytes integer.
9-12	“WAVE”	Format chunk marker (includes trailing null).
17-20	16	Length of format data as listed above.
21-22	1	Type of format (1 is PCM) - 2 bytes integer.
23-24	2	Number of Channels - 2 bytes integer.
25-28	44100	Sample Rate - 8 bytes integer.
29-32	88200	Byte Rate - 8 bytes integer.
33-34	2	Block Align (bytes per samples times number of channels).
35-36	16	Bits per sample - 2 bytes integer.
37-40	“data”	“data” chunk header. Marks the beginning of the data section.
41-44	File size	Size of the data section.

Table 2.1: WAVE header structure.

Chapter 3

Qt Audio Classes Overview

Qt Multimedia offers a range of audio classes, covering both low and high level approaches to audio input, output and processing. Given the aims of this project, the focus was on the classes for low level audio playback and recording.

The **QAudioOutput** class offers raw audio data output, while **QAudioInput** offers raw audio data input. Both classes have adjustable buffers and latency, so they are suitable for both low latency use cases and high latency (like music playback).

The low level audio classes can operate in two modes – *push* and *pull*. In *pull* mode, the audio device is started by giving it a **QIODevice**. For an output device, the **QAudioOutput** class will pull data from the **QIODevice** (using **QIODevice::read**()) when more audio data is required. Conversely, for *pull* mode with **QAudioInput**, when audio data is available then it will be written directly to the **QIODevice**. In *push* mode, the audio device provides a **QIODevice** instance that can be written or read to as needed. This results in simpler code but more buffering, which may affect latency.

QAudioDeviceInfo Class

QAudioDeviceInfo provides information about the audio devices, such as sound cards and USB headsets, that are currently available on the system. A **QAudioDeviceInfo** is used by Qt to construct classes that communicate with the device, such as **QAudioInput** and **QAudioOutput**.

It is possible to query each device for the formats it supports. A format in this context is a set consisting of a specific byte order, channel, codec, frequency, sample rate, and sample type. A format is represented by the **QAudioFormat** class.

QAudioFormat Class

The **QAudioFormat** class stores audio stream parameter information. It contains parameters that specify how the audio sample data is arranged. These are the frequency, the number of channels, the sample size, the sample type, and the byte order.

QAudioInput Class

The **QAudioInput** class provides an interface for receiving audio data from an audio input device. The **QAudioInput** is created with a specific **QAudioDeviceInfo** and with the **QAudioFormat** in which the incoming audio data will be encoded.

QAudioOutput Class

The **QAudioOutput** class provides an interface for sending audio data to an audio output device. It is constructed the same way as the **QAudioInput**, providing the **QAudioDeviceInfo** and the format of the outgoing data.

QIODevice Class

The **QIODevice** class is the base interface class of all I/O devices in Qt.

Chapter 4

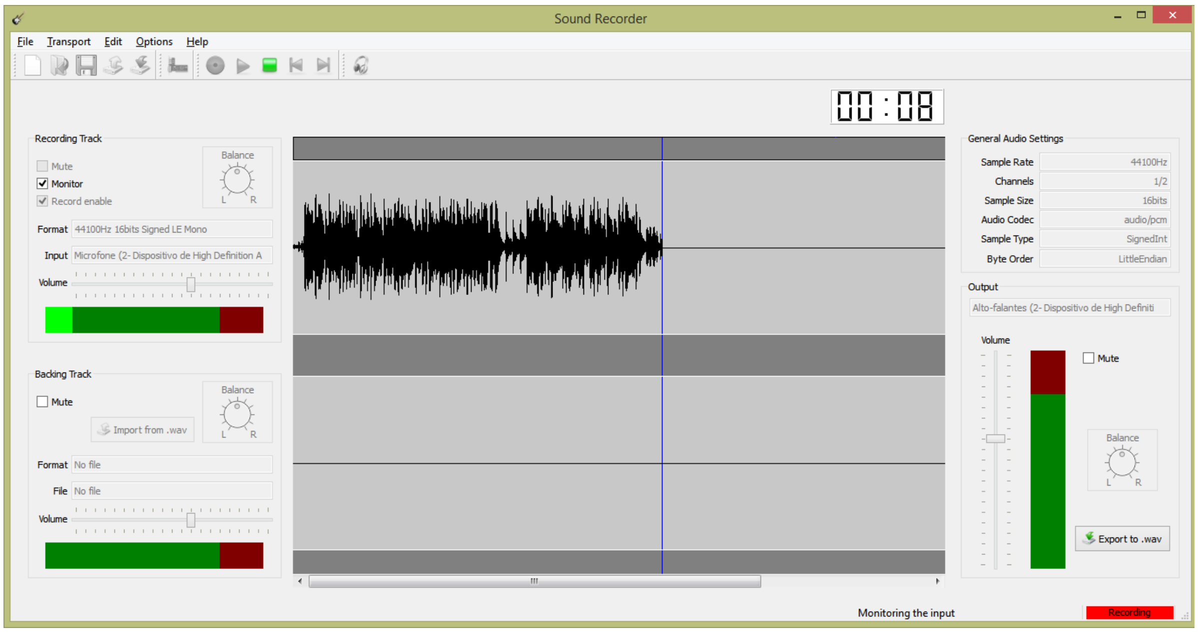
Functionalities

This chapter presentes the functionalities and features of ***SoundRecorder***.

4.1 The GUI

The GUI was built for displaying two soundtracks, one for Recording and one for the Backing Track. This last can be any WAVE file following the specification given in the section 2.2.

The Figure 4.1 shows the complete user interface while recording a track:



Recording Track: Mute ☐, Monitor ✓, Record enable ✓

Format: 44100Hz 16bits Signed LE Mono

Input: Microfone (2- Dispositivo de High Definition A

Volume

Backing Track: Mute ☐

Format: No file

File: No file

Volume

Sample Rate	44100Hz
Channels	1/2
Sample Size	16bits
Audio Codec	audio/pcm
Sample Type	SignedInt
Byte Order	LittleEndian

Output: Alto-falantes (2- Dispositivo de High Definiti

Mute ☐

Figure 4.1: GUI while recording

4.2 Menus and Toolbars

The menus and actions are shown in the Figure 4.2:

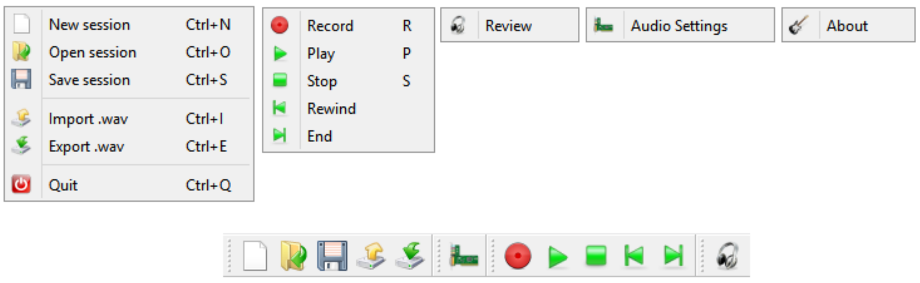


Figure 4.2: Menus and Toolbars of SoundRecorder

Not all the functionalities were implemented for this submission. Below, a brief explanation on each button:

File menu
- New session: Future implementation.
- Open session: Future implementation.
- Save session: Future implementation.
- Import .wav: Import a WAVE file to the Backing Track.
- Export .wav: Export the soundtrack after reviewing.
- Quit: Exit from the application.

Transport menu
- Record: Record audio in the Recorder Track and play the Backing Track.
- Play: Play the available audio data.
- Stop: Stop playing/recording.
- Rewind: Set the cursor to the beginning of the timeline.
- End: Set the cursor to the end of the longest track.

Edit menu
- Review: Disable recording, and prepare the audio for exporting.

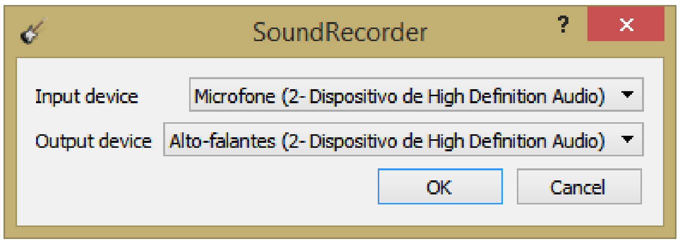
Options menu
- Audio Settings: Open the Settings Dialog for choosing the IO devices.

Help menu
- About: Open a simple “About” window.

Depending on the status of the sound engine, some buttons may be enabled or disabled.

4.3 The Settings Dialog Box

The Settings Dialog is shown in the Figure 4.3. This is the first place to go when launching the application. It allows to select the devices for input and output. The choice will be done between all the available devices on the computer.



SoundRecorder

Input device: Microfone (2- Dispositivo de High Definition Audio)

Output device: Alto-falantes (2- Dispositivo de High Definition Audio)

Figure 4.3: Settings Dialog

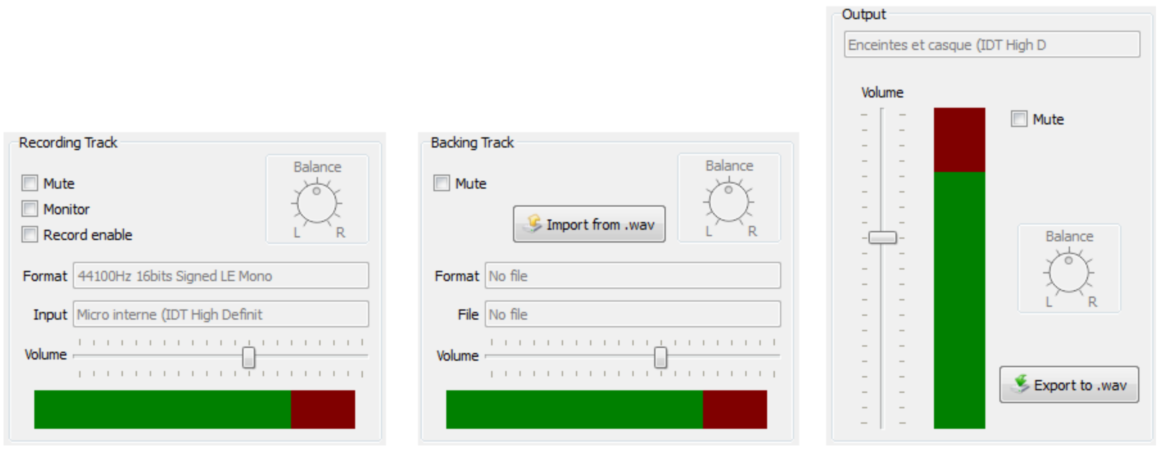
Therefore, for the application to work properly, it is necessary at least:

- One internal or external input (microphones, soundcards).
- One output (speakers, headphones or soundcards).

By clicking in "OK", the audio devices in the application will be initialized.

4.4 Recording Track, Backing Track and Output group boxes

The three group boxes, presented in Figure 4.4 are the interface needed to manage their respective track.



Recording Track

Format: 44100Hz 16bits Signed LE Mono

Input: Micro interne (IDT High Definit

Backing Track

Format: No file

File: No file

Output

Enceintes et casque (IDT High D

Figure 4.4: Tracks and Output group boxes

The output group box is used only when reviewing or editing after recording.

The **Mute** checkbox are used to mute the track while playing. In the case of the Output mute, it put all volumes to zero.

The **Record Enable** check box in the Recording Track allows to start recording by pressing the red button Record, or with pressing the key "R".

The **Monitor** check box enables monitoring the sound in the audio input.

The volume sliders are individual as the Mute checkboxes.

The information regarding the audio format and the IOs are displayed.

4.5 The Waveform Scroll Area

This central part of the window is the waveform of the sound, shown in detail in the Figure 4.5. By clicking on the darkest zone is possible to move the cursor through the timeline. There is a scroll bar in at the bottom for navigating in the waveform.

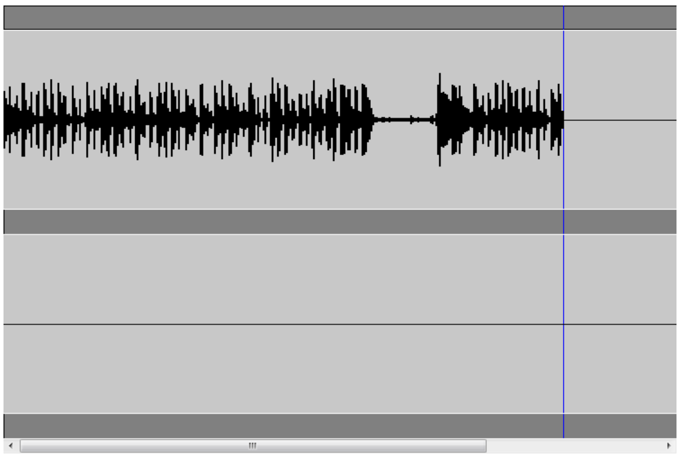


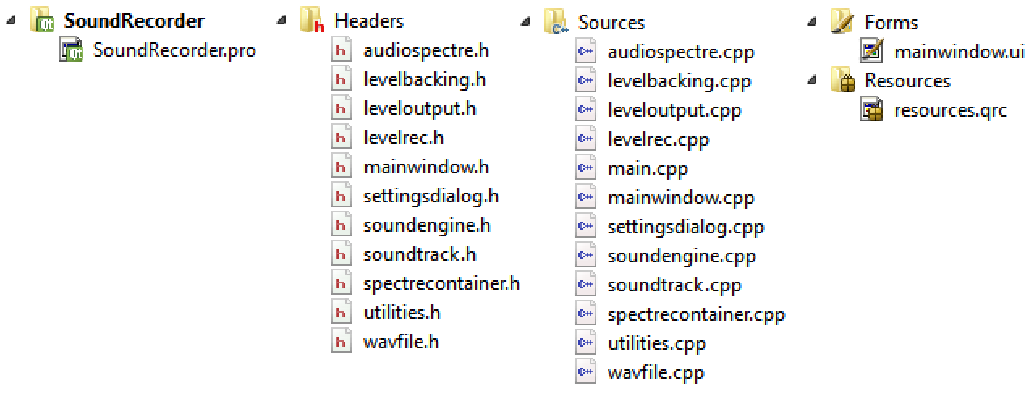
Figure 4.5: Waveform scroll area

Chapter 5

Implementation

5.1 Project Overview

In this chapter, the design and implementation of the classes will be addressed. The Figure 5.1 shows all files contained in the project directory:



```
SoundRecorder
    SoundRecorder.pro
Headers
    audiospectre.h
    levelbacking.h
    leveloutput.h
    levelrec.h
    mainwindow.h
    settingsdialog.h
    soundengine.h
    soundtrack.h
    spectrecontainer.h
    utilities.h
    wavfile.h
Sources
    audiospectre.cpp
    levelbacking.cpp
    leveloutput.cpp
    levelrec.cpp
    main.cpp
    mainwindow.cpp
    settingsdialog.cpp
    soundengine.cpp
    soundtrack.cpp
    spectrecontainer.cpp
    utilities.cpp
    wavfile.cpp
Forms
    mainwindow.ui
Resources
    resources.qrc
```

Figure 5.1: Project directory

The qmake project file contains the name of all the files in the project: headers, sources, forms and resources. But it also provides information to **qmake** to link it to the corresponding C++ libraries. In this application, qmake needs to get the libraries from the **core gui** (by default) and also from **mutlimedia**, which allows the use of all the **QtMultimedia** libraries in the project without the need of including it in each file.

5.2 Classes overview

The classes can be grouped as follows:

- MainWindow class
- Classes for managing and handlind sound (SoundTrack, WavFile and SoundEngine)
- Classes for displaying sound intensity (LevelRec, LevelBacking and LevelOutput)
- Classes for drawing the waveform (AudioSpectre and SpectreContainer)

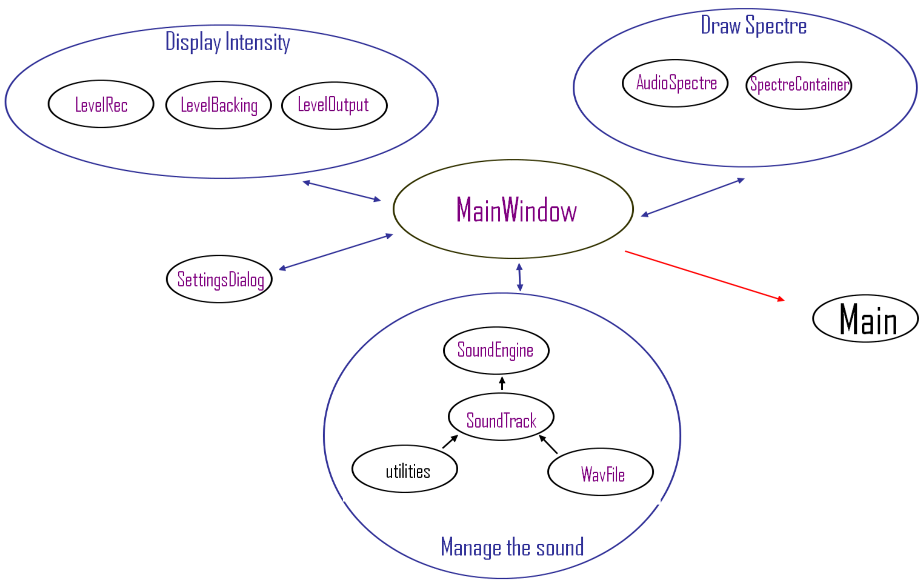


Figure 5.2: Classes relations

MainWindow Class

The **MainWindow** class is public derived from **QMainWindow**, which provides a framework for building an application's user interface. The object from this class is instantiated within the `main.cpp` source file, and then shown. It incorporates all the functionaties of the GUI and triggers the functions in SoundEngine and SettingsDialog classes.

LevelRec, LevelBacking and LevelOutput Classes

These classes, derived from QWidget, were built for controlling the three intensity bar widgets in MainWindow.

- **LevelRec** receives and draws the level value of the sound being played or recorder by the object of **SoundTrack** initialized as *RecorderTrackMode*.
- **LevelBacking** receives and draws the level value of the sound being played by the object of **SoundTrack** initialized as *BackingTrackMode*.

- **LevelOutput** receives and draws the level value of the sound being played by the SoundEngine in the editing mode.

SpectreContainer and AudioSpectre Classes

These two classes were designed for drawing the sound waveform in the MainWindow. They receive the timeline position and the intensity of the sound for drawing the waveform.

SettingsDialog Class

The **SettingsDialog** class, derived from **QDialog**, is responsible for launching the dialog window in which the IO devices are selected. It is triggered by the **MainWindow**, more precisely by the settings button. The information about the audio IO is then sent to the **SoundEngine**, for initialization of the audio devices. This class was inspired by the Spectrum Example [5], in the documentation of Qt Multimedia.

WavFile

This class is used for retrieving header information from the imported WAVE file and for writing new WAVE files from RAW data. This class is available in the documentation of Qt Multimedia, in the Spectrum Example [5].

SoundTrack

The **SoundTrack** class is responsible for dealing with inputs and outputs, recording, playing and monitoring the audio. Two objects of this class are instantiated in **SoundEngine**. One for the Recorder Track and one for the Backing Track.

SoundEngine

The **SoundEngine** class manages the objects of **SoundTrack** class, triggering functions and receiving signals. It is also in charge of controlling the timeline position and comprises the editing and exporting tasks.

Chapter 6

Drawing the Waveform

Creating the container

The waveform drawing can be done with a simple **QWidget** and a **paintEvent**. The problem is that the duration of sound data being recordeded is unknown and it is not possible to keep on drawing when you are going outside of the window. The solution implemented shown in the Figure 6.1:

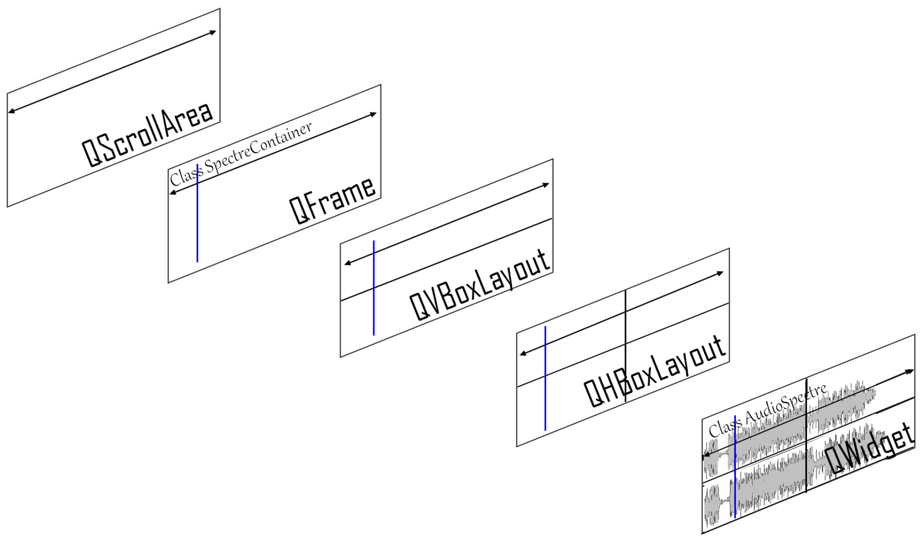


Figure 6.1: Waveform displaying scheme

Using **QScrollArea** is a good way to display widgets with no predefined size. Once the widget is inside, this one grows and the scroll bar of this QScrollArea becomes smaller.

The **QFrame** inside is the container for all widgets which will be added. This one has been promoted to an object from the **SpectreContainer** class. The **SpectreContainer** class is a simple QFrame object with one more attribute: **cursor position**. The cursor is drawn inside and it will be used to navigate in the waveform across the timeline.

The **QVBoxLayout** included inside the container will help to add two separated waveforms: one for the recorded/recording track, on the top and another for the backing track, in the bottom.

The **QHBoxLayout** is another layout, horizontal, which will permit to extend the waveform. The first widget is created already created in the beginning. When filled, another one is added to the QHBoxLayout, and the waveform is continuously drawn. Once a new widget is added, the QFrame inside the QScrollArea grows and we can scroll back further.

The **QWidget** is the final link in the chain. This one is promoted as object from the **AudioSpectre** class, to draw the spectre. It contains a **qreal** array of the size of the widget's width, to draw a bar for each intensity along the widget. This array is filled each time the buffer has been processed while recording or after importing.

Drawing while recording

Each time a buffer is filled, a level is coming through a signal. The **AudioSpectre** class will use this signal to get the level for filling it's array. In testing the **SoundEngine** status, a cell of the array is set when recording. Once the first cell of the array is filled, the cursor in **SpectreContainer** is moved the new coming level fills the second cell of the array, etc. When the cursor reaches the widget side, a new dynamic **AudioSpectre** is created and it starts being filled following the cursor.

Drawing when importing

When importing, there is no signal coming from the buffer of the QAudioInput. So a new buffer is created to process the whole new file. A loop calls the function which returns the level for each position in the file until it reaches the end. For each level, the **AudioSpectre** array is filled, as previously.

Chapter 7

SoundTrack Class

As mentioned before, the **SoundTrack** class is responsible for dealing with inputs and outputs, recording, playing and monitoring the audio. Two objects of this class are instantiated in SoundEngine. One for the Recorder Track and one for the Backing Track. This class has the SoundEngine class as friend.

Ahead is an overview of the main (not all) functions and object members within this class.

7.1 Functions, slots and signals

- Mutator **setMode(Mode mode)**
  Sets the mode in which the SoundTrack object will be initialized (**Mode** is a list).
- Public function **void loadFile(const QString &fileName)**
  Triggered by the QFileDialog in the MainWindow, it will analyze and open the WAVE file, when the SoundTrack object is initialized as *BackingTrackMode*.
- Private function **void initialize(const QAudioDeviceInfo &inputDeviceInfo, const QAudioDeviceInfo &outputDeviceInfo)**
  This function is called by the SoundEngine class, which pass the two parameters above that correspondes to the chosen audio IO devices. If the SoundTrack is initialized as *RecorderMode*, it will immediately create the audio objects. If the SoundTrack is initialized as *BackingTrackMode*, it will retrieve the information about the audio IO devices and wait until a WAVE file is loaded. Then, the audio objects will be created accordingly with the format of the WAV file. Actually, the only parameter that could be changed, in this project, is the number of channels. All the other parameters must be as shown in the section 2.2, otherwise the file will not be loaded and an error message will be shown.
- Private slot **void readMore()**
  This slot is triggered at each notification interval of QAudioOutput. At each *50 ms*, an amount of data is read from the file, the volume is applied, and it is sent to the

output. The first notification event is forced, and then, after *50 ms* a signal will be emitted by QAudioOutput. Right before writing in the output, the content of the buffer is sent to the function **calculateLevel()** for calculation of the normalized peak level which will be displayed in the intensity bar and in the waveform. The choice of the interval is related to the buffer size, the sample rate, the bit-depth and the number of channels, as shown in the section 2.2.

- Private slot **void calculateLevel(const QByteArray &levelBuffer, const qint64 &samples)**
  This slot is triggered after applying volume to the samples, inside **readMore()**. It performs the calculation of the normalized level, considering an amount of samples equal to the buffer size. This level is used in the intensity bars and in the waveform.

- Signal **void sendLevel(const qreal &level)**
  This signal is emitted at each Notification Interval, and contains the normalized level of a certain amount of bytes (buffer size) calculed by the **calculateLevel(const QByteArray &levelBuffer, const qint64 &samples)** function.

7.2 Object members

- **QAudioFormat** object
  Stores the audio format used in the SoundTrack.

- **QAudioDeviceInfo** object
  Stores the chosen audio hardware or driver.

- **QByteArray** object
  Temporarly stores audio data read from the file from applying volume and level calculation.

- **QIODevice** *object
  Base interface between hardware and QAudioInput or between QAudioOutput and hardware.

- **QAudioInput** *object
  Interface between QIODevice and audio data.

- **QAudioOutput** *object
  Interface between audio data and QIODevice.

Chapter 8

SoundEngine Class

As mentioned in the section 5.2, the **SoundEngine** class manages the objects of **SoundTrack** class, triggering functions and receiving its signals. It is also in charge of controlling the timeline position and comprises the editing and exporting tasks.

Following is an overview of the main (not all) functions and object members within this class.

8.1 Functions, slots and signals

- Mutator **setStatus(Status status)**
  Change the status in which the SoundEngine currently (**Status** is a list).
- Public slots **void setAudioInputDevice(const QAudioDeviceInfo &device)** and qtvoid setAudioOutputDevice(const QAudioDeviceInfo &device)
  These functions receive the chosen IO devices from the SettingsDialog and initialize the objects of the SoundTrack class, as well as the editing/exporting output within SoundEngine.
- Private function **void initializeAudio()**
  Act as a linker between the SettingsDialog and the initialization of the SoundTrack objects.
- Public slots **void stop()**, **void play()**, **void record()** and **void play()**
  Functions triggered by the GUI to execute basic transport tasks. The SoundEngine receives the signals from MainWindow and the call the specific functions members of the objects of class SoundTrack.
- Public slot **void updateTimeLinePosition()**
  Receive the signal emitted at each audio notification interval and processes the timeline recording/playing position.

- Signal **void timeLinePosition(const qint64 &time)**
  Emit the timeline position which is used by the MainWindow for displaying the waveform and the timeline cursor.

8.2 Object members

- **SoundTrack** *recorderTrack
  Object responsible for recording and playing audio.
- **SoundTrack** *backingTrack
  Object responsible for playing the imported WAVE file.

Chapter 9

Conclusion

This project served as a solid introduction to object-oriented programming and it has provided us with a solid and intense interaction and practicing with Qt. It was very challenging and we feel more comfortable now with the contents of the course than we felt before.

In the beginning, it was hard to get good results. It took us time for a complete understanding of Qt's Multimedia Classes and the task of drawing the waveform has proved to be a little more complicated than we thought.

The Qt's documentation was very helpful in all the stages of the development. Some examples were taken as reference or inspiration, mainly the ones provided in Qt Multimedia Examples [4].

At this level of C++ learning, it was very interesting to have the possibility to propose a project instead of having a compulsory one. Even though our chosen theme has no direct relation with Computer Vision, it was a really good starting point.

References

[1] Burg, Jennifer; Romney, Jason; Schwartz, Eric; *Digital Sound & Music* (http://csweb.cs.wfu.edu/ burg/CCLI/Documents/Chapter5.pdf).

[2] IBM Corporation and Microsoft Corporation; *Multimedia Programming Interface and Data Specifications 1.0*.

[3] *Qt Documentation, Qt Multimedia 5.4, Audio Overview* (http://doc.qt.io/qt-5/audiooverview.html).

[4] *Qt Documentation, Qt Multimedia 5.4, Qt Multimedia Examples* (http://doc.qt.io/qt-5/multimedia-examples.html).

[5] *Qt Documentation, Qt Multimedia 5.4, Spectrum Example* (http://doc.qt.io/qt-5/qtmultimedia-spectrum-example.html).

Appendix A

Trello's board

Trello was used as project management tool during the development of the project. Below are the links for the Bi-weekly reports and for the board itself.

- Trello's Board C++ Project - SoundRecorder
- Bi-weekly report 1 - March 20th
- Bi-weekly report 2 - April 3rd
- Bi-weekly report 3 - April 17th
- Bi-weekly report 4 - May 1st