2016

Architecture overview

MASTER AUDIO TECHNOLOGY FUNCTIONS

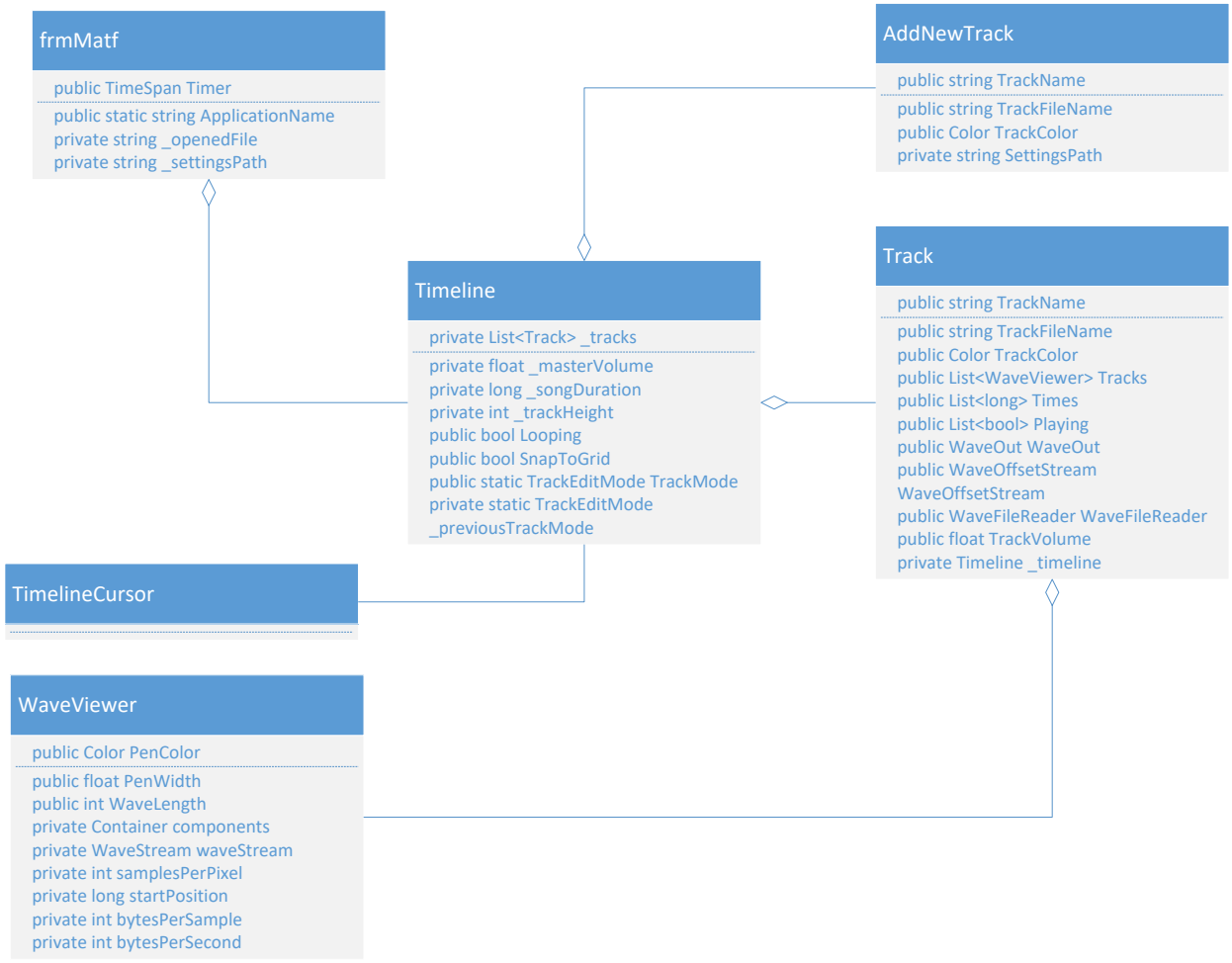
CVETKOVIĆ VANJA, DIMITRIJEVIĆ PREDRAG, NENADOVIĆ ĐURO

FACULTY OF MATHEMATICS, BELGRADE | Studentski Trg 16, Belgrade

Contents

Program organisation

Project is organised in its main classes. There are six main classes and each of them represents some part of GUI and implements its behaviours. Below is the diagram of organisation.



Organisational diagram

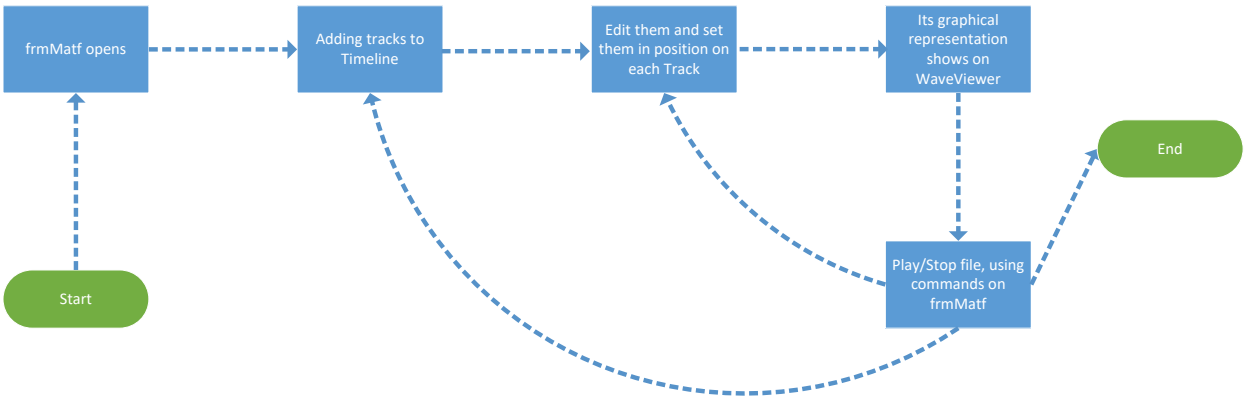
As we can see "frmMatf" class is the main GUI class and everything starts from there. From there, we go to class "Timeline" which implements timeline for audio files, container for its graphical representation and playing. "AddNewTrack" gather all info needed for adding audio file, "Track" contains the audio file and plays it, and "WaveViewer" gives us graphical representation of audio file. "TimelineCursor" gives us graphical cursor to track progress of the files we are streaming.

There are several alternatives, one of them is to gather all classes in one, which is hard to maintain, other is not to separate viewers from timelines, but they are all hard to maintain because we need to allow app to grow with adding of functionalities, so good distribution of work between classes is crucial. Also, we were considering more partitioning of app but at this level this is optimal organisation.

Classes

We've described classes above, but let's look at them with more details.

Basic workflow could be as described below:



Basic workflow

Now, let's look at every class description.

frmMatf

This class is the main GUI class for containing all others. When the program starts this is the first thing we see.

Timeline

Timeline is container for holding tracks, its graphical representation and manipulating it.

AddNewTrack

This class is collecting new file info for main app to read it. This is new window but it depends on Timeline.

Track

Track is holding info about audio file. It can play it, and do many other manipulations with it.

TimelineCursor

It is the element that is showing us progress of play flow.

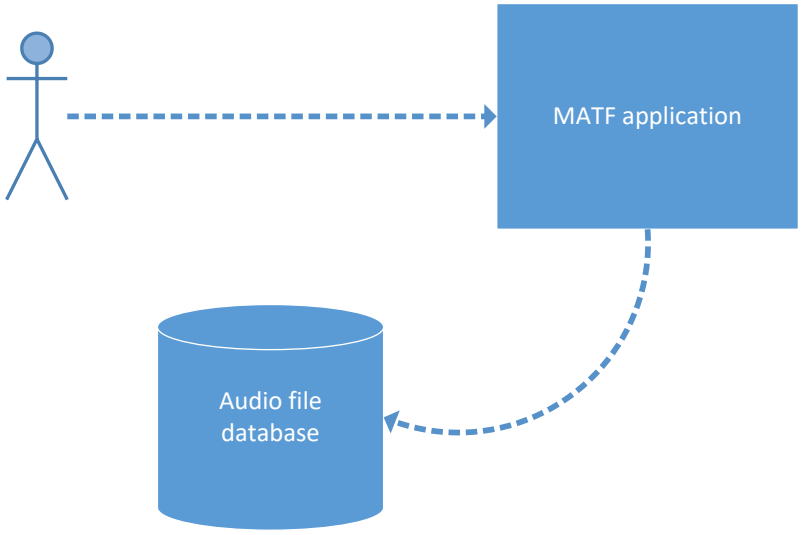
WaveViewer

This class allows us to see graphical representation of audio file.

This organisation of work proved to be the most efficient, as we have taken in consideration many other, but none of them showed better results.

Data design

The main database that we are using is simple folder, with its subfolders, divided in categories, filled with audio file samples. Using this base of samples, we can build complex audio files.



Simple data flow chart

This kind of data organisation is good because we can easily add data, no matter which format, type, size, name or anything else is: if format is supported, we can add it. If we had a different kind of database, it would be tough to maintain such amount of data, and that would influence the performance.

Business rules

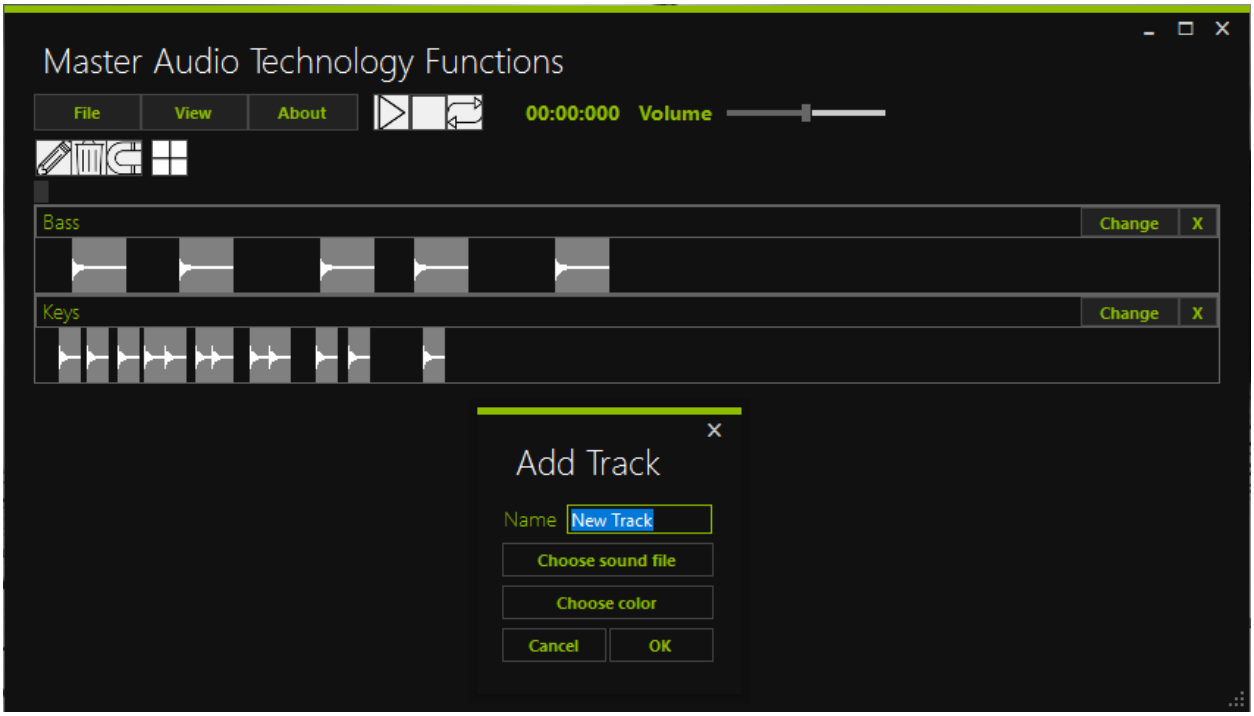
We had no special kind of business rules, but only thing we needed to follow is for code to be open, free to use, distribute and contribute. In that way we can rest assure that app is going to grow and become even more powerful and reliable.

User interface

Graphical user interface is shown below.

As we can see, now all class names and functions becomes clear: “frmMatf” is the main window, “Timeline” is the container in which are placed “Tracks”, and “AddNewTrack” dialog is for adding new files. “WaveViewer” has added tracks and gave them graphical representation.

As it can be seen, it’s intuitive, easy to use, and easy to learn.



Graphical user interface

Input/output

We've talked about reading data above, but here we're just review main concepts.

Reading schema is "just-in-time", as we are reading file and reproducing it in real time. We start reading it as reproducing starts. But it's not all that simple, as we read data using schema "look-ahead" to plot its graphical representation.

Error detecting is on level of whole audio file: if audio file is in wrong format or damaged, we throw error, and do not add it.

Resource management

Main resources we are using is memory. CPU usage is low, but we need to constantly read from memory in order to reproduce sound.

Though it may sound tough for OS to maintain stability while using so much memory, app creates a small buffer in which it reads parts of audio files which then reproduces, and therefore reduces need for memory, as we are not keeping files in memory, instead we're reading it from disc in a real time.

Safety and performance

Coding convention set some rules in writing a code, so we ended up in very efficient code.

Everything is working smoothly and in real time, which is very important for user, not to have to wait for program to return to its working condition.

Scalability and interoperability

Working on a scalability is the main thing for app which intends to grow.

Main structure is developed and a lot of changes implemented with no need for changing structure. The main thing is that structure is divided in separate units, so every feature can be added as part of them, or as independent part.

As a result, we can say that this app is highly scalable.

We do not expect from our system to share data, although our database is totally free to use anywhere.

Localization/internationalisation

For now, only English language is supported. We have a plan of adding new supported languages.

Error management and stability

All errors that we encountered are maintained and processed.

We can say that this app is highly stable.

Architecture feasibility

This kind of architecture can be done, in all planned limits, as analysis shows.

Changing strategy

Using a SCRUM methodology we are able to respond to changing of strategy, and architecture is designed in that way so we can change various things, quickly.