**The Audio Programmers**

**SERQET**

**Analysis Class Report**

***Revision History***

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Authors | Description of Change | Sections | Rev | Date |
| Elias | Improved saw wave efficiency |  |  | 2-12-19 |
| Robert | Implemented Oscillator for sound generation |  |  | 2-12-19 |
| James | Implemented Low pass filter |  |  | 2-18-19 |
| Alex | Implemented slider and slider behavior skeleton |  |  | 2-25-19 |
| Elias | Added triangle wave |  |  | 2-27-19 |

**Table of Contents**

[1 Team Description](#_30j0zll) 3

[2 Grammatical Parse](#_1fob9te) using Project Description 3

[2.1 Analysis Class Diagram](#_3znysh7) 4

[2.2 Analysis Class List](#_2et92p0) 5

[2.2.1 Main Component](#_3dy6vkm) 5-6

[2.2.2](#_1t3h5sf) Delay 7

# Team Description

|  |  |
| --- | --- |
| **Team Member Name** | **Email Address** |
| Alexander Feldman | feldman0007@csu.fullerton.edu |
| James Dobson | bobberino98@csu.fullerton.edu |
| Elias Jackson | eliasthemusician@csu.fullerton.edu |
| Robert Rivas | r\_rivas@csu.fullerton.edu |

# Grammatical Parse using Project Description

SERQET is an audio processing application that will accept MIDI input from the user to control the pitch of the oscillators in a synthesizer. MIDI (Musical Instrument Digital Interface) is a standard protocol for communicating musical information in digital music production. The synthesizer will be a playable instrument, intuitively designed, with a suite of effects to enhance the sound.

*Detailed Description of SERQET:*

Our application will consist of a simple GUI interface with various buttons, knobs, and sliders that will allow users to customize the sound to their liking. Our program will process the MIDI input, using it to control the synthesizer, and then modify the output sound with effects. Effects will include reverb, delay, distortion in order to give the user the widest array of sounds possible. The user will also be able to change the waveform of the oscillators, and use filters to alter the sound of the oscillators.

*Potential Class General Classification*

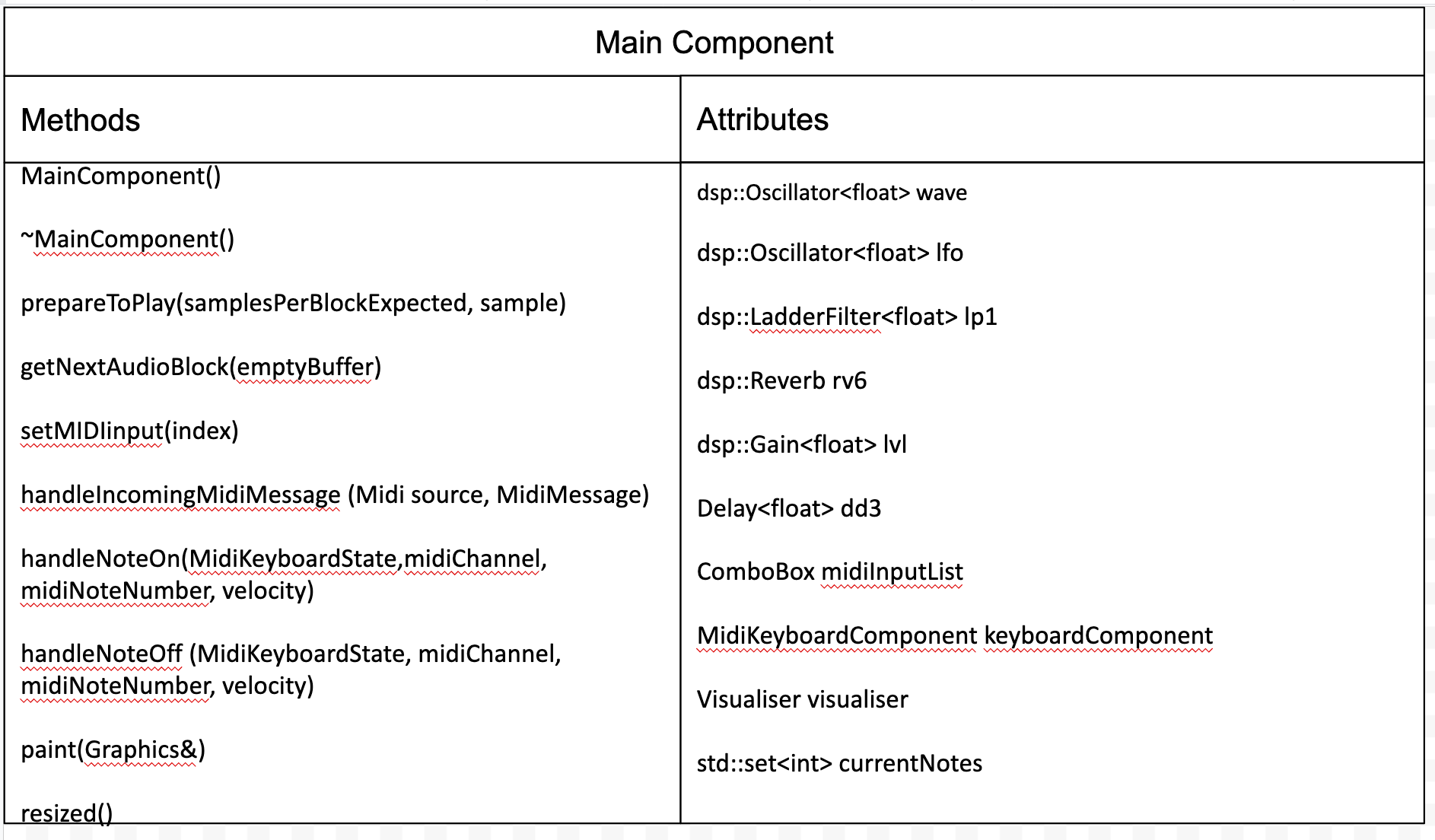
Controller External entity

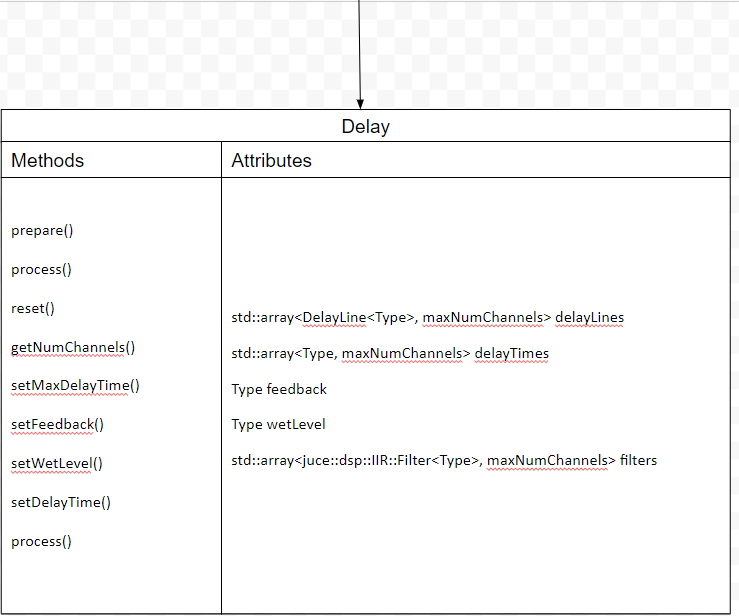
Wave Form organizational unit

Main Component Structure/User Interface (Main GUI component)

Effects Control

## Analysis Class Diagram





## Analysis Class List

|  |  |
| --- | --- |
| **Sequence Number** | **Analysis Class** |
| 1 | Main |
| 2 | Controller |
| 3 | Effects |
| 4 | Wave Form |

### <Main Component>

Description: Handles construction of user interface, including visible sliders and keyboard

Methods:

* MainComponent()
  1. Draws user interface
  2. Draws visualizer
  3. Establishes audio connection
* ~MainComponent()
  1. Shuts down the audio device and clears the audio source
* prepareToPlay()
  1. Called when the audio device is started, or settings are changed
  2. Calls other preparation functions for oscillator and effects
* getNextAudioBlock()
  1. Processes waveforms and effects based on parameters
  2. Only processes when there are keys being held down
* setMIDIinput()
  1. Called when clicking on a MIDI device
  2. Sets the current device
* handleIncomingMidiMessage()
  1. Sets variables that signify MIDI input, so other functions work correctly
* handleNoteOn()
  1. Sets the frequency of the current note whenever the keyboard is pressed
  2. Adds this note to the currentNotes variable
* handleNoteOff()
  1. Upon releasing a note, removes the note from currentNotes
* paint()
  1. Fills background color
* resized()
  1. Handles UI rearrangement upon window resizing

Attributes:

* DSP stands for Digital Signal Processing. The “dsp” namespace is imported from the JUCE library and handles most of the audio generation.
* dsp::Oscillator<float> wave: This class generates sine waves, saw waves, and square waves.
* dsp::Oscillator<float> lfo: An LFO that we use to modulate the filter cutoff.
* dsp::LadderFilter<float> lp1: A user-adjustable filter that cuts off sound above a certain frequency.
* dsp::Reverb rv6: Reverb effect.
* dsp::Gain<float> lvl: Allows for control of the gain level.
* Delay<float> dd3: Delay effect.
* ComboBox midiInputList: List of available MIDI devices.
* MidiKeyboardComponent keyboardComponent: Displays an on-screen keyboard which controls the synth.
* Visualiser visualiser: An object that displays a waveform visualizer.
* std::set<int> currentNotes: A set that stores the current notes being held down.

### <Delay>

Description: Handles the processing of the delay effect.

Methods:

* Delay(): Constructor for the delay class, initializes it to default values
* prepare(): Prepares the class to start processing audio
* reset(): Resets the dlay, clearing the delay buffers
* getnumChannels(): Returns the number of audio channels being processed
* setMaxDelayTime(): Sets the maximum time (in seconds) that the delay can be
* setFeedback(): Sets the amount of repeqats allowed by the delay
* setWetLevel(): Sets the volume of the delays
* setDelayTime(): Sets the length of the delay (in seconds)
* process(): Takes in audio and processes the delay effect

Attributes:

* std::array<DelayLine<Type>, maxNumChannels> delayLines: An array of all delay line buffers being used in the class.
* std::array<Type, maxNumChannels> delayTimes: An array of the delay times for each delay line
* Type feedback: the variable for the amount of repeats in the delay
* Type wetLevel: The variable for the volume of the delay
* std::array<juce::dsp::IIR::Filter<Type>, maxNumChannels> filters: An array storing the filter for each delay (each delay has a low pass filter applied to it);