МИНИСТЕРСТВО ЭНЕРГЕТИКИ, ПРОМЫШЛЕННОСТИ И СВЯЗИ СТАВРОПОЛЬСКОГО КРАЯ

**Государственное бюджетное**

**профессиональное образовательное учреждение**

**«Ставропольский колледж связи**

**имени Героя Советского Союза В.А. Петрова»**

Цикловая комиссия вычислительной техники

УТВЕРЖДАЮ

Заместитель директора по учебной работе

\_\_\_\_\_\_\_\_\_\_\_\_ Г.А. Белоусова

«\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_2019 г.

**ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА**

09.02.03 Программирование в компьютерных системах

Создание эквалайзера на C#

|  |  |
| --- | --- |
|  | Выполнил: Тащук М.В. |
|  | 4 курс, группа П 153 |
|  | Руководитель: Артемов С.В. |
| Зав. отделением | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |
|  | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_И.В. Черкасова |
| Председатель ЦК | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_И.В. Еремина |
|  |  |

Ставрополь, 2019

**Содержание**

[Введение 3](#_Toc11082365)

[1 Эквалайзер и bass.dll 4](#_Toc11082366)

[1.1 Основные понятия 4](#_Toc11082367)

[1.2 Работа эквалайзера 7](#_Toc11082368)

[1.3 Bass.dll 8](#_Toc11082369)

[1.4 Bass.NET 9](#_Toc11082370)

[2 Эквалайзер EqPlayer 13](#_Toc11082371)

[2.1 Архитектура эквалайзера 13](#_Toc11082372)

[2.2 MainWindow.cs 14](#_Toc11082373)

[2.3 BassPlayer.cs 19](#_Toc11082374)

[2.4 Eq.сh 23](#_Toc11082375)

[3 Интеграция модулей 27](#_Toc11082376)

[3.1 Базовые требования 27](#_Toc11082377)

[3.2 Интеграция BassPlayer.cs 28](#_Toc11082378)

[3.3 Интеграция eq.cs 32](#_Toc11082379)

[Заключение 36](#_Toc11082380)

[Список литературы 37](#_Toc11082381)

[Приложение 38](#_Toc11082382)

# Введение

Все чаще социальные сети и крупные организации создают свои собственные оконные приложения. Некоторая часть этих организаций вводит возможность прослушивания музыки. А при разработке приложения на C# это может стать проблемой. Чаще всего языки разработки приложений не предусматривает активные аудио проигрыватели. Для этого была создана библиотека bass.dll.

Создание отдельных программных модулей представляет собой сложный процесс изучения физического объекта, требующего реализации, описание его свойств и функций. Иногда это занимает много времени и сил, но позволяет тысячам других разработчиков экономить на этом время.

Цель выпускной квалификационной работы: разработка интегрируемого программного модуля эквалайзера

Задачи:

* изучить теоретические основы разработки программных модулей;
* провести анализ средств разработки программных модулей;
* создать программный модуль;
* провести отладку и тестирование программного модуля;

Выпускная квалификационная работа состоит из введения, трех глав, заключения, списка литературы и приложения.

# 1 Эквалайзер и bass.dll

## **Основные понятия**

Во время обзора проекта будут встречаться термины, которые требуют предварительного объяснения. Это необходимо для понимания логики и схемы работы программы.

**Эквалайзер** (англ. **equalize** - "выравнивать", общее сокращение - "**EQ**"), темброблок - это устройство или компьютерная программа, которая позволяет увеличивать, либо уменьшать громкость отдельных зон частотного диапазона, выравнивать амплитудно-частотную характеристику звукового сигнала, то есть корректировать его (сигнала) амплитуду избирательно, в зависимости от частоты.

Частота дискретизации звука — это количество измерений уровня сигнала за 1 секунду. Герц (Гц) или Hertz (Hz) — это научная единица измерения, определяющая количество повторений какого-то события в секунду.

Например, когда мы говорим, что частота дискретизации 44,1 кГц, то это значит, что сигнал измеряется 44 100 раз в течение одной секунды. На рисунке.1. показано изменение сигнала косинусоидального сигнала, при частоте дискретизации 1900Гц и на рисунке видно что сигнал изменяется 40 раз за 0.02 секунды.

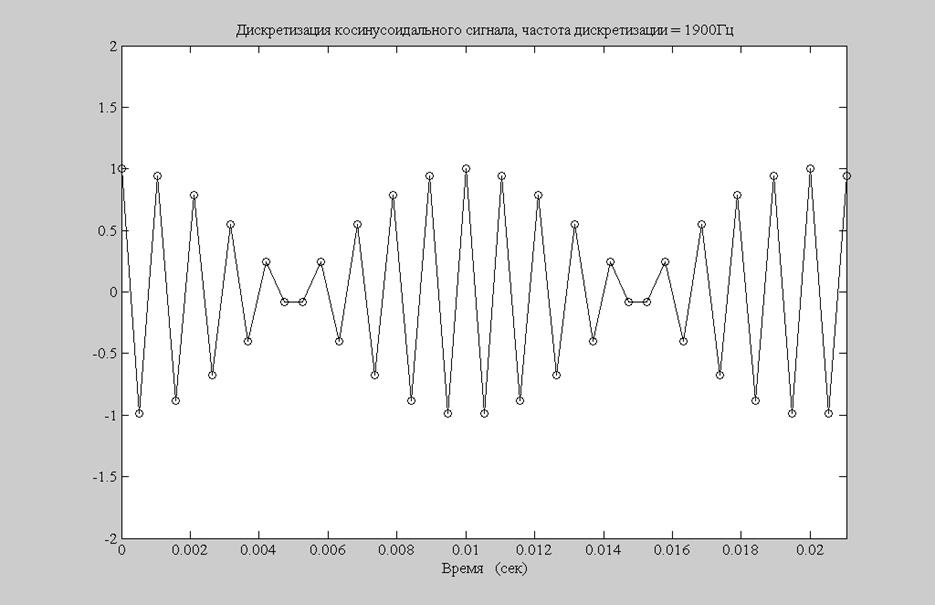


Рис.1. Частота дискретизации 1900Гц за 0.02с

Основной вопрос на первом этапе преобразования аналогового сигнала в цифровой (оцифровки) и последующего его вывода на проигрывание состоит в выборе частоты дискретизации аналогового сигнала. Чем больше частота, тем точнее соответствует цифровой сигнал аналоговому.

Частота звука – это количество гормональных повторений в секунду. Гормональные повторения – это колебания, при которых физическая величина изменяется с течением времени по гармоническому (синусоидальному, косинусоидальному) закону.

Таким образом, амплитуда звуковой волны изменяет значение амплитуды (громкости) в секунду с количеством раз, равному частоте данного звука. Частота человеческого голоса равна от 300 до 3500 герц, и именно из-за такого частого колебания волны, мы не слышим промежутков между ними. Человеческое ухо способно распознать эти колебания при звуке до 150 герц, все частоты выше будут казаться одним беспрерывным звуком.

Эквалайзер изменяет не только отдельно взятую частоту, а также затрагивает ближе стоящие частоты, диапазон которых определяется добротностью эквализационной полосы. Таким образом в частотной характеристике появляется не просто частотная яма, а плавный диапазон который выравнивается перед следующей эквализационной полосой. Плавность изменения диапазона с помощью эквалайзера Fab Filter Pro Q показана на рисунке.2.



Рис.2. Fab Filter Pro с добротностью Q = 1

Изменение амплитуды высоких и средних частот малой полосой добротности, чаще всего приводит к резонансам, возникающими из-за сильного превосходства одних частот над другими. Такая эквализация называется «колоколообразной» или «bell» из-за сходства таких резонансов с ударами колокола.

Peaking(полосовые) фильтры или Bell(колоколообразующие) фильтры, это полосы находящиеся между крайними значениями фильтра и изменяют громкость самого сигнала. Фильтр назван полосовым из-за его расположения в активном частотном спектре, и может изменять громкость отдельно взятых диапазонов, определенных добротностью полосы эквализации.

На рис.3. показан пример изменения амплитуды частот при помощи полосового фильтра.

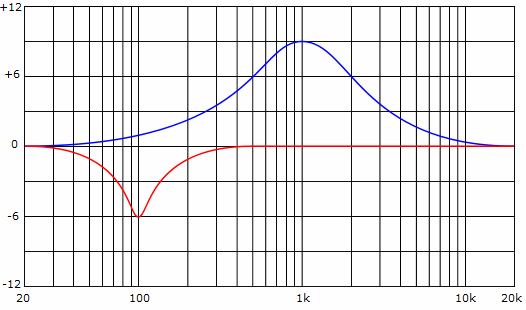


Рис.3. Изменение амплитуды звучания частотного диапазона при помощи полосового фильтра

## **Работа эквалайзера**

Принцип работы эквалайзера основан на воздействии на определенную часть звукового спектра, в результате которого увеличивается или уменьшается амплитуда звучания частот на данном диапазоне, то есть их громкость.

Работая с графическими эквалайзерами, чаще всего используется всего два параметра – **Freq**, который определяет центральную частоту фильтра и **Gain**, который определяет коэффициент усиления на центральной частоте фильтра, то есть коэффициент увеличения громкости. К этому списку можно добавить еще выбор типа фильтров эквалайзера, но практически во всех современных программных эквалайзерах этот выбор происходит автоматически и зависит от первоначального места размещения узла на частотном диапазоне.

Наравне с параметрами центральной частоты и коэффициента усиления фильтров, существует еще один крайне важный параметр – Bandwidth (добротность фильтров) (**Q**), который определяет ширину усиливаемой или ослабляемой полосы частот. Проще говоря, чем выше значение добротности, тем уже полоса частот, и чем ниже значение добротности, тем она шире. Все это, в первую очередь, касается колоколообразных фильтров.

## **1.3 Bass.dll**

Bass.dll это мультиплатформенная библиотека для создания аудио потоков и управления ими. Определение устройства вывода позволяет библиотеке преобразовывать цифровые аудиофайлы в аналоговый сигнал, и воспроизводить на любом аудиоустройстве. Цифровой звук устроен таким образом, что каждый воспроизводимый в данный момент файл имеет свой собственный поток, который следует на проигрывающее устройство, и уже там воспроизводится вместе с остальными активными потоками.

Bass.dll библиотека от организации Un4seen, позволяет создавать цифровые аудио потоки, и выводить их на устройства воспроизведения. Для ее использования на официальном сайте требуется получить цифровую подпись, которая указывает легальность использования данной библиотеки. Одиночные разработчики и малые компании могут получить данную подпись бесплатно, но для крупных компаний ее получение будет стоить от 125 долларов.

Вначале воспроизведения каждого файла, создается поток, подключенный к данному файлу. После завершения проигрывания поток удаляется. Огромное множество аудио потоков могут создавать большие приложения, такие как видеоигры.

Bass.dll это отличная библиотека для создания аудио проигрывателей. Тот контроль, который получает пользователь над потоками, позволяет реализовать любые необходимые проигрывателю свойства. Возможность добавлять эффекты и контролировать их внутри программы неоценимо переплетается с малыми требованиями.

Все что необходимо для создания проигрывателя это наличие самой библиотеки, и больше никаких пристроек или программ не нужно. В этом и есть главное преимущество bass.dll, для реализации всех возможностей, достаточно иметь стандартную платформу разработки. Это позволяет создавать аудио потоки с минимальной нагрузкой на память, а также уменьшить количество классов и функций, что позволяет существенно уменьшить временные затраты на разработку.

Также мультиплатформенность bass.dll позволяет вести разработку на любом языке программирования. Набор шестнадцатеричных данных, не привязан ни к одному языку, потому любой язык программирования может использовать эту библиотеку для создания и управления аудиопотоками.

## **1.4 Bass.NET**

Для интеграции разработки с bass.dll в Visual Studio было создано расширение bass.NET. Так как bass.dll является набором мультиплатформенных операций, то для использования их в .NET фреймворке организацией radio42 была создана еще одна библиотека получившая название bass.NET. Так как bass.dll не может напрямую подключаться к проекту на фреймворке .NET, bass.NET связывает функции из bass.dll и позволяет использовать их в ваших проектах.

Для подключения bass.dll к .NET фреймворку, в содержимое проекта необходимо добавить bass.dll, но в таком случае он просто так не заработает. Следующий шаг это подключение в ячейку ссылок bass.NET. И на этом установка библиотеки будет завершена, и необходимо будет только подключить ее к классу, в котором она будет реализована.

1. using Un4seen.Bass;

Перед тем как создавать поток, необходимо инициализировать библиотеку bass.dll. Инициализация необходима для определения устройства воспроизведения, и подключения к нему потоков. Для этого указывается логическая переменная инициализации устройства InitDefaultDevice, и логический оператор проверяет, была ли она уже инициализирована, и если нет, то проводит инициализацию.

BASS\_Init() является функцией инициализации устройства воспроизведения и первым параметром принимает устройство воспоизведения. Устройства воспроизведения по умолчанию имеют индекс -1, и это позволяет функции инициализироваться на любое устройство, работающее в данный момент. Второй параметр передает частоту дискретизации сигнала, и это единственный параметр который принимает внешняя функция. Третий параметр передает особые флаги устройства. С помощью данных флагов можно установить моно или стерео проигрывание, а так же можно установить воспроизведение в режиме 3D и уже далее устанавливать позицию и перемещения сигнала в пространстве. И последний параметр при установке Zero указывает, что используется оконное приложения.

1. public static bool InitDefaultDevice;
2. private static int HZ = 44100;
3. public static bool InitBass(int hz)
4. {
5. if (!InitDefaultDevice)
6. InitDefaultDevice = Bass.BASS\_Init(-1, hz, BASSInit.BASS\_DEVICE\_DEFAULT, IntPtr.Zero);
7. return InitDefaultDevice;
8. }

* 1. **Эквалайзер в Bass.dll**

Одним из преимуществ bass.dll является возможность установки звуковых эффектов на аудио поток. Он включает в себя внушительное количество эффектов которые используют звукорежиссёры по всему миру: компрессор, параметрический эквалайзер, хорус, флангер, эхо, а также функцию реверберации.

Данные эффекты используют технологию DirectX Media Object что подразумевает наличие установленного DirectX 8 или выше.

Система предлагает два пути установки эффектов на канал. С использованием флага BASS\_SAMPLE\_FX или же без него. Преимущества такого флага это возможность мгновенного применения эффектов на аудио поток, что в случае его отключения происходит медленнее. Но главным недостатком является возможность активации эффектов только при использовании Direct Sound вывода. И данный способ не может быть использован, так как модуль ориентирован на широкую аудиторию пользователей, а многие из них используют сторонние выводы аудио сигнала.

Эквалайзер в bass.dll называется BASS\_DX8\_PARAMEQ и DX8 используется в названии для обозначения того что эффект может работать только с DirectX 8 или выше.

Структура эквалайзера имеет три параметра. В отличии от главных функций проигрывания, модуль эффектов принимает числа в дробном формате. Причина тому использование DirectX Media Object, который не способен обрабатывать целочисленные переменные.

1. typedef struct
2. {
3. float fCenter;
4. float fBandwidth;
5. float fGain;
6. } BASS\_DX8\_PARAMEQ;

Первый параметр это значение центральной частоты полосы. Пиковая частота не может указываться более 16 000. Связанно это с тем, что частоты выше 16 000 Гц имеют слишком высокую частоту колебаний и не различаются человеческим ухом. Потому регулирование частот выше данного значения попросту не имеет смысла.

Второй параметр отвечает за добротность фильтра. Значения указываются в полутонах, и каждые двенадцать полутонов образует собой октаву. Каждая октава увеличивает свое значение частоты в два раза, таким образом, частота ноты ЛЯ первой октавы равна 440 Гц, а Ля второй октавы уже составляет 880 Гц.

Третий параметр является самой амплитудой волны, т.е громкостью. Максимальное отклонение значения составляет 15dB в обе стороны.

# 2 Эквалайзер EqPlayer

## **2.1 Архитектура эквалайзера**

Для разработки эквалайзер активно использовалось ООП. На рис.4. показана основная архитектура классов эквалайзера

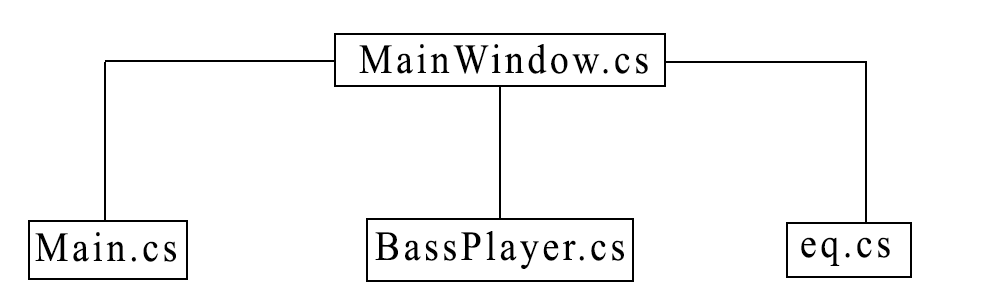


Рис. 4. Классовая архитектура эквалайзера

Так как целью проекта является создание независимого эквалайзера, каждый класс является отдельным объектом. Как и в реальном мире, аудио проигрыватель является программой подключаемой к главной станции, а эквалайзер это отдельный предмет который так же подключается к потоку сигнала и изменяет его.

Потому вся программа разделяется на 4 класса.

Первый класс MainWindow.cs является аналогом основной станции (компьютер, аналоговый синтезатор), к которой подключаются проигрыватели и периферия эффектов. В нем содержаться элементы управления проигрывания и эквализации. К нему подключается класс Main.cs который хратит в себе переменные основного класса.

BassPlayer.cs - это класс реализующий аудио проигрыватель. Он содержит в себе функции создания аудио потоков и управления ими. Класс проигрывателя является аналогом проигрываемой программы, и выполняет свои функции при подключении к объекту основной станции.

Eq.cs - класс выполняющий функции аналогового эквалайзера. Он подключается к аудио потоку и через него проходит весь сигнал, отправляя на выход уже обработанный. Так как эквалайзер является устройством, имеющим входной сигнал, для него необходим аудио проигрыватель создающий поток этого сигнала, который в последствии будет подвержен эквализации.

На рис.5. показана схема работы программы, и каким образом происходит преобразование сигнала при прохождении программы.

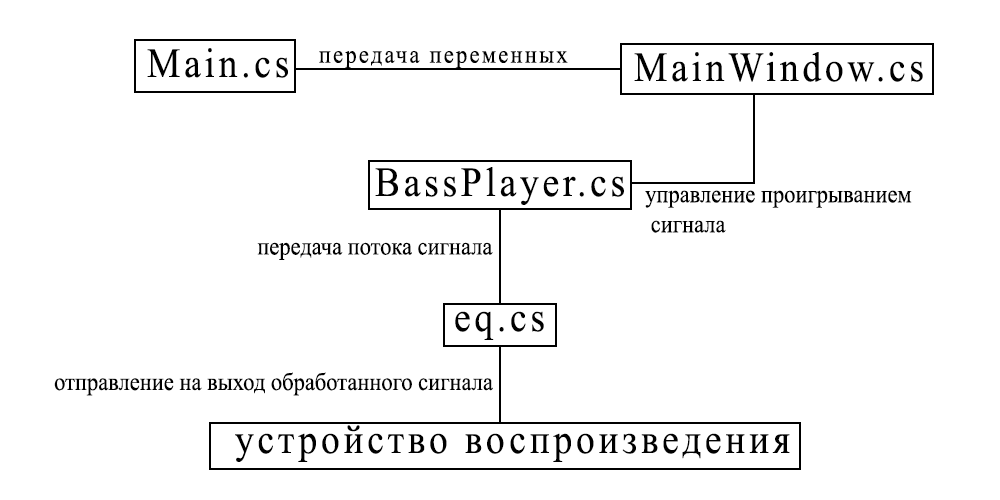


Рис.5. Процесс обработки сигнала

Первостепенно из класса Main.cs главная форма получает данные о проигрываемых файлов. Затем с помощью элементов управления, реализует функции класса проигрывателя, и после этого создает аудио поток, устанавливает его громкость и устройство вывода. Затем к этому потоку подключается эквалайзер, и пропускает этот сигнал через частотные фильтры, отправляя на выход уже обработанный сигнал. Затем этот сигнал достигает устройства воспроизведения и преобразуется в аналоговый сигнал.

Именно такой путь проходит аудио поток через всю программу.

## **2.2 MainWindow.cs**

Основное окно является управляющим элементом и хранит все основные элементы управления. Оно делится на две части. Первая часть, это аудио проигрыватель, который имеет основные элементы управления для воспроизведения аудио. Интерфейс плеера представлен на рисунке 5.

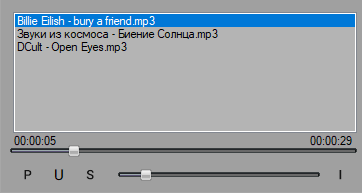


Рис.5. Интерфейс плеера

Интерфейс проигрывателя состоит из списка аудиозаписей (плейлиста), четырех кнопок управления и два ползунка прогресса.

Для реализации интерфейса были использованы NuGet пакеты MaterialSkin v0.2.2 от Ignace Maes и Tools.Slider от MONTAUFRAY.

NuGet - это функция Visual Studio позволяющая устанавливать общедоступные пакеты парой кликов мыши. Так данное расширение уменьшает время поиска необходимых плагинов, так как может производить поиск по ключевым словам.

MaterialSkin v0.2.2 добавляет новые элементы в панель управления и имеет больше визуальных настроек. Использование данного расширения связано с созданием минималистичного и настраиваемого интерфейса. Каждый элемент MaterialSkin плагина имеет множество визуальных настроек, что позволяет создать любой дизайн.

Tools.Slider – это ползунок который используется для полосы прогресса и звука, а так же данные ползунки играют роль полос эквалайзера.

Плейлист хранит названия всех треков и позволят переключать их. Он работает за счет динамического списка, и может быть расширен или сокращен.

Верхний ползунок показывает прогресс воспроизведения и позволяет перейти к любому участку аудиозаписи, просто переместив его туда. Так же над ним располагаются таймеры, которые указывают на текущее и общее время воспроизведения.

Второй ползунок позволяет менять громкость воспроизведения.

Четыре кнопки являются управляющими элементами воспроизведения.

P – Play, кнопка активации воспроизведения, она активирует функцию создания потока, или же продолжает проигрывание если оно было приостановлено.

U – Pause, кнопка для приостановки воспроизведения и последующем его воспроизведении с того же места при нажатии кнопки P.

S – Stop, кнопка останавливающая воспроизведение, путем остановки и удаления текущего аудио потока.

I – Import, кнопка открывающая диалоговое окно, которое позволяет добавлять аудиозаписи в плейлист.

Также важным элементом данного проигрывателя является таймер.

Объект таймера необходим для обновления данных прогресса воспроизведения.

Кнопка Play запускает функцию BassPlayer.Play, который создает новый аудио поток, а так же добавляет информацию о проигрывании. Операции записи данных о треке в объекты trackTime, playTime, slTime занимает большую часть функции, но так же важный элемент здесь это активация таймера.

1. private void playButton\_Click(object sender, EventArgs e)
2. {
3. if((playList.Items.Count != 0) && (playList.SelectedIndex != -1))
4. {
5. string current = Main.\_files[playList.SelectedIndex];
6. BassPlayer.Play(current, BassPlayer.\_volume);
7. trackTime.Text = TimeSpan.FromSeconds(BassPlayer.GetTimeOfStream(BassPlayer.\_stream)).ToString();
8. playTime.Text = TimeSpan.FromSeconds(BassPlayer.GetPosOfStream(BassPlayer.\_stream)).ToString();
9. slTime.Maximum = BassPlayer.GetTimeOfStream(BassPlayer.\_stream);
10. slTime.Value = BassPlayer.GetPosOfStream(BassPlayer.\_stream);
11. timer.Enabled = true;
12. }
13. }

Период таймера настроен на 500 миллисекунд, потому функция timer\_Tick каждые пол секунды обновляет данные прогресса проигрывания. Функция TimeSpan преобразует секунды полученные из функции GetPosOFStream() в читаемый счетчик времени, разбирая его на часы и минуты.

slTime - это ползунок прогресса проигрывания. Он отображает положение текущего проигрывания по линии.

1. private void timer\_Tick(object sender, EventArgs e)
2. {
3. playTime.Text = TimeSpan.FromSeconds(BassPlayer.GetPosOfStream(BassPlayer.\_stream)).ToString();
4. slTime.Value = BassPlayer.GetPosOfStream(BassPlayer.\_stream);
5. }

Кнопка Stop помимо вызова соответствующей функции, так же останавливает счетчик таймера. И возвращает данные прогресса в нулевое положение.

1. private void stopButton\_Click(object sender, EventArgs e)
2. {
3. BassPlayer.Stop();
4. timer.Enabled = false;
5. slTime.Value = 0;
6. playTime.Text = "00:00:00";
7. }

Графический эквалайзер имеет 10 полос эквализации, и каждая имеет подпись соответствующей ей частоты. Отличия графического эквалайзера от параметрического заключается в том, что полоса нее имеет возможности сменить частоту и добротность фильтра. Каждое значение фиксировано, и это позволяет изменять амплитуду, не задевая другие полосы. Дизайн эквалайзера представлен на рисунке 6.

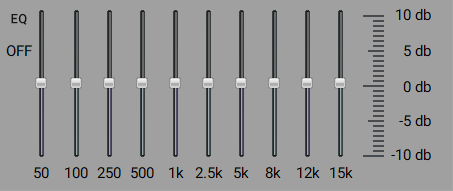


Рис.6. Дизайн эквалайзера.

Справа от полос имеется график изменения амплитуды частот. Максимальное и минимальное значение нельзя опустить ниже 10 дб.

Активация эквалайзера происходит по нажатию кнопки EQ и текстовое поле под ней показывает состояние эквалайзера. Переменная EqIsActive хранит в себе состояние активации, и в зависимости от нее вызываются функции активации и дезактивации.

1. private void eqButton\_Click(object sender, EventArgs e)
2. {
3. if (eq.EqIsActive == false)
4. {
5. eq.ActivEq(Main.eqValues, BassPlayer.\_stream);
6. eq.EqIsActive = true;
7. eqTextSwitch.Text = "ON";
8. } else
9. {
10. eq.DeactivEq();
11. eq.EqIsActive = false;
12. eqTextSwitch.Text = "OFF";
13. }
14. }

В классе Main.cs имеется массив чисел равных текущим значениям полос эквализации. И после изменения каждой из полос, эти значения обновляются, а затем отправляются в функцию которая обновляет значения эквалайзера.

1. private void sl15khz\_Scroll(object sender, EventArgs e)
2. {
3. Main.eqValues[9] = sl15khz.Value - 10;
4. if (eq.EqIsActive == true)
5. eq.UpdateEq(Main.eqValues);
6. }

Таким образом при изменении значения любой из полос, эквалайзер обновляет данные всех полос и применяет их к сигналу, при условии что эквалайзер активирован.

## **2.3 BassPlayer.cs**

Класс BassPlayer представляет собой виртуальную копию проигрывающего устройства, в которое заложены функции управления воспроизведения. Так как класс содержит только функционал, для его реализации потребуется создать интерфейс управления и привязать к его элементам функции класса.

Первым пунктом является объявление переменных необходимых для работы.

Переменная SR (Sample Rate) содержит значение частоты дискретизации, о которой шла речь ранее. Она необходима для инициализации аудио потока.

InitDefaultDevice – это логическая переменная, отображающая состояние инициализации устройства воспроизведения.

\_stream – целочисленная переменная, содержащая код потока, и в будущем преобразуется в указатель потока. Ссылаясь на данный указатель, функции получают доступ к управлению потоком.

\_volume – содержит значения громкости сигнала.

1. private static int SR = 44100;
2. public static bool InitDefaultDevice;
3. public static int \_stream;
4. public static int \_volume = 100;

Первая функция InitBass() необходима для первичной инициализации устройства воспроизведения. В случае если устройство не инициализировано, выполняется функция BASS\_Init() речь о которой шла в разделе 1.4.

1. public static bool InitBass(int hz)
2. {
3. if (!InitDefaultDevice)
4. InitDefaultDevice = Bass.BASS\_Init(-1, hz, BASSInit.BASS\_DEVICE\_DEFAULT, IntPtr.Zero);
5. return InitDefaultDevice;
6. }

Функция активации проигрывания получает в качестве параметров путь к файлу, который необходимо воспроизвести.

Первым этапом выполняется проверка было ли приостановлено проигрывание, используется функция BASS\_ChannelIsActive для получения значения состояния работы потока и сравнивается с флагом BASSActive.BASS\_ACTIVE\_PAUSED.

Затем применяется функция Stop() для очистки и удаления потока. Предыдущий поток необходимо удалить по двум причинам. Первая – для каждого аудиофайла должен быть выделен отдельный аудио поток, но создавать множество потоков может доставлять проблемы. Потому требуется в первую очередь освободить переменную потока, а затем записывать в нее данные нового. Второй причина – это расход памяти. Каждый аудио поток несет в себе в первую очередь полную информацию о файле, размер которых иногда может достигать сотен мегабайт. И в дополнение к этому, так же требуется память для записи флагов, обработки функций и применению эффектов. Потому создание множества аудио потоков порой может расходовать слишком много памяти.

После инициализации запускается функция создания нового аудио потока. BASS\_StreamCreateFile() создает аудио поток, загружая в него данные аудиофайла, для дальнейшего его воспроизведения. Первым параметром является полный путь к файлу. В классе Main.cs имеется массив содержащий полные пути всех импортированных файлов. Этот массив форматируется, отделяя названия от полного пути, и отправляет данные в контейнер типа ListBox, в котором все объекты отображаются в той же последовательности, что и в массиве, и это позволяет воспроизводить их через индекс выделенного файла.

Второй параметр указывает отступ, с которого будет воспроизводиться аудиофайл. Третий параметр указывает длину воспроизведения потока. Если оба этих параметра установлены на ноль, то файл будет воспроизводиться от начала до конца.

Четвертый параметр устанавливает флаги воспроизведения. Такие же флаги используются при инициализации устройства воспроизведения, о котором речь шла выше.

При успешном создании аудио потока, функция SetVolumeToStream() устанавливает громкость канала, в соответствии с установленной переменной.

И последняя функция BASS\_ChannelPlay() запускает проигрывание канала, и отправляет сигнал на выход.

1. public static void Play(string filename, int vol)
2. {
3. if (Bass.BASS\_ChannelIsActive(\_stream) != BASSActive.BASS\_ACTIVE\_PAUSED)
4. {
5. Stop();
6. if (InitBass(SR))
7. {
8. \_stream = Bass.BASS\_StreamCreateFile(filename, 0, 0, BASSFlag.BASS\_DEFAULT);
9. if (\_stream != 0)
10. {
11. SetVolumeToStream(\_stream, \_volume);
12. Bass.BASS\_ChannelPlay(\_stream, false);
13. }
14. }
15. }
16. else
17. Bass.BASS\_ChannelPlay(\_stream, false);
18. }

Канальные функции в bass.dll позволяют вытягивать данные непосредственно из аудио потока. Это позволяет получать или устанавливать позицию текущего воспроизведения в реальном времени.

Функция GetPosOfStream() возвращает количество секунд прошедших с момента начала проигрывания. Для этого используется функция BASS\_ChannelGetPosition() которая возвращает бинарное число, которое впоследствии преобразуется в секунды с помощью функции BASS\_ChannelBytes2Seconds().

Данная функция необходима для установки позиции ползунка прогресса, а также времени проигрывания. Для получения всего времени, которое будет играть аудио поток, используется похожая функция, но вместо метода BASS\_ChannelGetPosition() используется BASS\_ChannelGetLenght(). А затем повторяется метод преобразования бит в секунды.

1. public static int GetPosOfStream(int stream)
2. {
3. long pos = Bass.BASS\_ChannelGetPosition(stream);
4. int posSec = (int)Bass.BASS\_ChannelBytes2Seconds(stream, pos);
5. return posSec;
6. }

Для перемотки воспроизведения используется функция SetPosOfScroll() которая активируется при изменении значения ползунка прогресса, и второй параметр принимает позицию равному значению ползунка.

1. public static void SetPosOfScroll(int stream, int pos)
2. {
3. Bass.BASS\_ChannelSetPosition(stream, (double)pos);
4. }

Громкость воспроизведения регулируется функцией SetVolumeToStream(). Она принимает значение потока, и громкости. Этот параметр связывается с объектом который регулирует громкость. И при его изменении в функцию передается новое значение громкости, записывает его в переменную, и устанавливает на аудиопоток.

Функция BASS\_ChannelSetAttribure() помимо аудио потока, принимает значение типа атрибута, который должен быть изменен. В данном случае для изменения громкости используется атрибут BASS\_ATTRIB\_VOL. Последнее значение является самой величиной назначаемой громкости. В связи с тем, что в аудио проигрывателях громкость указывается в процентах, в данном модуле оно так же указывается значением от 1 до 100. Но сам атрибут принимает дробное значение от 0 до 1. Для этого происходит деление полученного числа на 100.

1. public static void SetVolumeToStream(int stream, int vol)
2. {
3. \_volume = vol;
4. Bass.BASS\_ChannelSetAttribute(stream, BASSAttribute.BASS\_ATTRIB\_VOL, \_volume / 100F);
5. }

## **2.4 Eq.сh**

Возможность регулировании амплитудной характеристики частот, обычным пользователям дается крайне редко. В основном такую функцию содержат некоторые аудиоплееры, или специальные программы. Но такие программы как правило изменяют частоты не только аудио, но и всех системных звуков, что может быть проблемно.

В подавляющем количестве приложений, с возможностью проигрывания аудио, нет функции эквалайзера. Связанно это с тем, что для его интеграции необходимо обращаться к модулям на другом языке, и их интеграция занимает намного больше времени и ресурсов. Именно эту проблему призван решить модуль Eq.ch.

Eq.ch это модуль, который могут использовать другие разработчики в своих приложениях. Он содержит все необходимые функции для подключения и настройки эквалайзера. На рисунке 7 представлена схема работы эквалайзера.

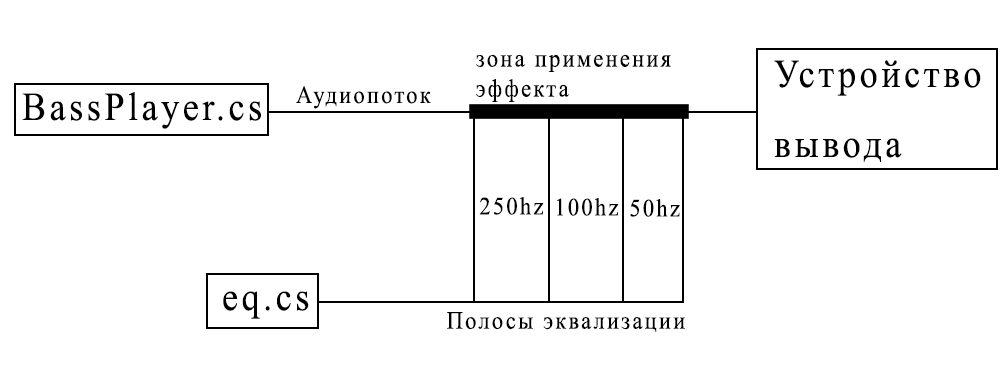


Рис.7. Схема работы эквалайзера

Первым делом создается экземпляр класса эквалайзера, именно этот класс хранит параметры центральной частоты, добротности и амплитуды. Это действие объявляет присутствие эквалайзера, и что он будет влиять на аудио поток. В данный экземпляр будут записаны параметры экранизации, а так же именно он будет влиять на сигнал потока.

1. public static BASS\_DX8\_PARAMEQ EQ = new BASS\_DX8\_PARAMEQ();

Функция ActivEq() инициализирует значения, и пускает поток через обработку эквалайзера.

Первым пунктом идет определение значения добротности. Оно зависит от количества полос эквалайзера. Так как модуль является интегрированным, его можно привязывать к любому количеству полос эквализации, которое пожелает разработчик. Но для верного баланса изменения частот, если количество полос не превышает пятнадцати штук, добротность будет равна 12 полутонам. Если число полос превышает 15, то ширина приравнивается к половине октавы.

Следующим запускается цикл инициализации значений эквалайзера.

Перед их выполнением требуется объявить несколько целочисленных массивов.

Первый массив fx будет хранить в себе объекты эффектов, для последующей передачи данных в экземпляр эквалайзера.

Массив freq хранит значения центральных частот. Данные числа определяют к каким частотам будет привязан эффект эквализации.

eqValues это массив значений амплитуды каждой полосы эквалайзера. Он связан с ползунками громкости и принимает значение при изменении одного из ползунков, а затем передает это значение в стек эффектов.

1. public static int[] fx = new int[] { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 };
2. public static int[] freq = new int[] { 50, 100, 250, 500, 1000, 2500, 5000, 8000, 12000, 15000 };
3. public static int[] eqValues = new int[] { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 };

Массив fx становится контейнером значений эквалайзера, а каждый ее элемент - управляющей ячейкой. При помощи функции BASS\_ChannelSetFX() эти значения становятся модулирующими объектами потока. Принимая первое значение потока, оно присоединяет эквализационную полосу к аудио потоку. Из-за того что экземпляр эквалайзера не может быть напрямую привязан к сигналу, используется такой «проводник».

Второй параметр указывает флаг эффекта, который будет применен к данной ячейке. Для каждого типа эффектов должен быть строго определен флаг. Во время присвоения данного типа, ячейка формируется таким образом, чтобы записать данные именно этого эффекта для последующей ее передачи.

Третий же параметр определяет приоритет применения этого эффекта. Так для создания последовательной цепи обработки из разных эффектов, например «компрессор-эквалайзер-фэйзер», для компрессора необходимо указать приоритет 2, эквалайзер получает значение 1 и фэйзер составит нулевой приоритет.

После инициализации массива эффектов, он привязывается к экземпляру эквалайзера с помощью функции SetParametersEQ().

1. public static void ActivEq(int[] eqFx, int[] fx, int[] freq, int handle)
2. {
3. if (fx.Length < 15)
4. bandwidth = 12.0;
5. else
6. bandwidth = 6.0;
7. for (int i = 0; i < fx.Length; i++)
8. {
9. fx[i] = Bass.BASS\_ChannelSetFX(handle, BASSFXType.BASS\_FX\_DX8\_PARAMEQ, 0);
10. SetParametersEQ(fx[i], freq[i], eqFx[i]);
11. }
12. }

SetParametersEQ() записывает значения добротности, центральной частоты и громкости в экземпляр класса эквалайзера. Затем функция BASS\_FXSetParameters() складывает значения выбранной структуры, преобразуя ее в числовой формат, и записывает в ячейку эффекта, которая в свою очередь привязана к аудио потоку.

Именно для этого и был необходим флаг эквалайзера, чтобы подготовить ячейки к записи данных в формате выбранного эффекта. После того как запись одной ячейки завершена, цикл переходит к следующей. И так каждая ячейка становится полосой эквализации, привязанной к аудио потоку.

1. public static bool SetParametersEQ(int fx , int center, int gain)
2. {
3. EQ.fBandwidth = bandwidth;
4. EQ.fCenter = (float)center;
5. EQ.fGain = (float)gain;
6. return Bass.BASS\_FXSetParameters(fx, EQ);
7. }

Функция UpdateEQ предназначена для смены параметров при изменении значения одной из полос эквализации. В функцию установки параметров передаются данные измененной полосы, и ее порядковый номер p, начиная с нуля. Затем функция применяет новые данные в ячейку эффектов.

1. public static void UpdateEq(int[] eqFx, int[] fx, int[] freq, int p)
2. {
3. SetParametersEQ(fx[p], freq[p], eqFx[p];
4. }

Для отключения эквалайзера требуется очистить все элементы массива эффектов. Так как ячейки эффектов напрямую подключаются к потоку, единственная возможность ее отключения, это очистка всех параметров.

Для этого используется функция DeactivEq(). Она принимает значение массива эффектов и при помощи цикла, используя функцию удаления эффектов с канала, удаляет параметры всех ячеек массива.

1. public static void DeactivEq(int[] fx)
2. {
3. for (int i = 0; i < fx.Length; i++)
4. Bass.BASS\_ChannelRemoveFX(BassPlayer.\_stream, fx[i]);
5. }

Класс разрабатывался как интегрируемый компонент. Любой разработчик создающий приложение на C# в состоянии добавить его в свою программу. Изучение библиотеки может занять много времени, и при сжатых сроках разработки это попросту невозможно.

# 3 Интеграция модулей

## **Базовые требования**

Интеграция модулей это процесс внедрения функций программного модуля в рабочий проект. Это необходимо для сокращения времени разработки.

Существуют целые компании создающие программные модули выполняющие отдельные функции, и продает их другим организациям. И заинтересованные лица с радостью платят за готовый программный код, который всего лишь надо внедрить в проект.

Именно такими модулями являются BassPlayer.cs и eq.cs. Использование данных модулей позволяет избежать необходимости в изучении библиотеки bass.dll и сэкономить много времени.

Модули разработаны с такой структурой, что не требуют изменений при внедрении в любую программу. Так как модули являются отражением реально существующих объектов, они имеют достаточно полный функционал основных параметров. Вся передача данных должна исходить из других классов программы, которые созданы для хранения переменных.

Такая схема разработки модулей является более предпочтительной, ведь идея использование готового кода заключается в том, чтобы не тратить время на изучение библиотек или алгоритмов.

Независимость BassPlayer.cs и eq.cs позволяет интегрировать их как вместе, так и по отдельности. Таким образом eq.cs можно интегрировать в уже написанный плеер под bass.dll.

Так же плюс данных модулей заключается в их малых требованиях. Для плеера единственным требованием является наличие библиотеки bass.dll, а для эквалайзера в дополнение должен идти DirectX 8 или выше.

Если разрабатывается оконное приложение, то потребуется сама графическая форма в которой будут находиться элементы управления, и еще один класс, в котором будут содержаться переменные и некоторые функции.

## **3.2 Интеграция BassPlayer.cs**

Модуль BassPlayer.cs содержит операции запуска, приостановки и остановки проигрывания, а также операцию импорта. Предпочтительно назначение данных операций на соответствующие кнопки.

Для отображения прогресса проигрывания необходимо установить таймер. Который будет останавливаться во время операций приостановки и остановки воспроизведения.

Также если это не консольное приложение следует поставить в форму объект типа ListBox который будет выполнять функцию плейлиста, и объект файлового диалогового окна, для импорта выбранных аудиозаписей в плейлист.

Ползунок потребуется для отображения и контроля прогресса проигрывания. Второй ползунок будет отвечать за громкость воспроизведения.

Пара надписей для отображения времени проигрывания. Первый будет отображать текущее время воспроизведения, второй покажет все время проигрывания.

Первая функция, которая срабатывает при нажатии клавиши импорта, открывает диалоговое окно, в котором затем выбираются необходимые аудиофайлы.

1. private void importButton\_Click(object sender, EventArgs e)
2. {
3. openFileDialog1.ShowDialog();
4. }

После того как были выбраны необходимые аудиофайлы, и нажата кнопка OK, функция сперва записывает файлы в строчный массив \_files полные пути к файлам.

1. public static List<string> \_files = new List<string>();

После записи файлов в массив, их необходимо отразить в плейлисте. Для этого используется функция класса Main.cs GetFileName();

1. private void openFileDialog1\_FileOk(object sender, CancelEventArgs e)
2. {
3. Main.\_files.Add(openFileDialog1.FileName);
4. playList.Items.Add(Main.GetFileName(openFileDialog1.FileName));
5. }

Функция GetFileName() принимает строчный параметр пути файла, а затем разделяет его функцией Split разделительным символом которых является косая черта. Так как косая черта в C# отвечает за экранирование символов, приходится одну черту экранировать другой, именно поэтому в параметре находятся две косые черты.

Затем возвращается последний элемент получившегося массива, который и является конечным именем файла.

1. public static string GetFileName(string file)
2. {
3. string[] tmp = file.Split('\\');
4. return tmp[tmp.Length - 1];
5. }

После импортирования аудиофайлов, их можно воспроизводить. Делается это при помощи кнопки playButton. Перед запуском воспроизведения, требуется выбрать необходимый файл в плейлисте, тогда плеер готов к запуску.

Первым делом происходит проверка не является ли плейлист пустым. Для этого используется параметр Count объекта ListBox, и если он не равен нулю, то проверяется был ли выбран файл в списке.

Значение индекса ListBox по умолчанию равно -1. Такое значение указывает, что ни одна из строк не была выделена.

Запуск процесса воспроизведения без данных проверок может привести к ошибке, и остановке работы программы, потому они являются обязательными при создании плеера.

Затем создается строчная переменная в которую помещается путь к файлу. Так как порядок элементов массива и плейлиста одинаковый, это позволяет получать путь аудиофайла через индекс выбранного элемента плейлиста.

Затем вызывается функция Play модуля BassPlayer в которую передается путь выбранного файла. Алгоритм данной функции описан в разделе 2.4.

Следующие действия инициализируют параметры прогресса воспроизведения.

Объект trackTime является текстовым контейнером, и грубо говоря, указывает длину аудиофайла. Функция GetTimeOfStream() возвращает лишь суммарное количество секунд. Для отображения времени в формате ЧЧ.ММ.СС используется атрибут FromSeconds объекта TimeSpan. Затем для присвоения значения текстовому контейнеру, его необходимо преобразовать в строчный тип. Для этого предназначена функция ToString().

Чтобы ползунок прогресса двигался во время воспроизведения, указывая положение текущего воспроизведения, его максимально возможное значение приравнивается к общему количеству секунд. Используя туже функция GetTimeOfStream(), максимально возможному значению объекта slTime присваивается время всего воспроизведения. Затем его значение обновляется с помощью таймера.

Главной частью изменения прогресса воспроизведения является активация таймера.

1. private void playButton\_Click(object sender, EventArgs e)
2. {
3. if((playList.Items.Count != 0) && (playList.SelectedIndex != -1))
4. {
5. string current = Main.\_files[playList.SelectedIndex];
6. BassPlayer.Play(current);
7. trackTime.Text = TimeSpan.FromSeconds(BassPlayer.GetTimeOfStream(BassPlayer.\_stream)).ToString();
8. slTime.Maximum = BassPlayer.GetTimeOfStream(BassPlayer.\_stream);
9. timer.Enabled = true;
10. }
11. }

Для обновления данных прогресса используется функция таймера – Tick. Эта функция выполняется каждый раз, как проходит интервал таймера. Интервал указывается в миллисекундах, таким образом, указывая значение 500, функция будет выполняться каждые пол секунды. А при значении 10 000 – раз в 10 секунд.

Интервал таймера установлен на одну секунду, это обеспечивает корректное отображение объекта указывающего время текущего воспроизведения.

playTime как раз является таким объектом. Представляя из себя обычный текстовый контейнер, он повторяет те же действия что и trackTime, с той разницей, что используется функция GetPosOfStream() которая вместо всего времени воспроизведения, возвращает его текущую позицию.

Так как максимально возможному значению ползунка прогресса было присвоено общее число секунд, то изменение его текущего значения будет отображать прогресс, наглядно показывая его текущее положение как показано на рисунке 8.



Рисунок.8. Изменение ползунка прогресса, и текстового указателя

1. private void timer\_Tick(object sender, EventArgs e)
2. {
3. playTime.Text = TimeSpan.FromSeconds(BassPlayer.GetPosOfStream(BassPlayer.\_stream)).ToString();
4. slTime.Value = BassPlayer.GetPosOfStream(BassPlayer.\_stream);
5. }

Функция остановки воспроизведения, назначена на кнопку stop, и помимо вызова функции Stop модуля BassPlayer, останавливает таймер и обнуляет значения параметров прогресса.

1. private void stopButton\_Click(object sender, EventArgs e)
2. {
3. BassPlayer.Stop();
4. timer.Enabled = false;
5. slTime.Value = 0;
6. playTime.Text = "00:00:00";
7. }

Приостановка проигрывания временно прекращает проигрывание аудио потока, с возможностью продолжить проигрывание с того же места.

1. private void pauseButton\_Click(object sender, EventArgs e)
2. {
3. BassPlayer.Pause();
4. timer.Enabled = false;
5. }

Для перемотки аудиофайла или изменения громкости, на ползунках используется функция Scroll. Она срабатывает каждый раз когда на одном из ползунков изменяется значение. И в случае с ползунком прогресса, применяет функцию SetPosOfScroll() которая принимает аудио поток, и новое значение ползунка, перематывая воспроизведения потока. Так как максимальное значение равно всему времени проигрывания, не требуется никаких преобразований нового значения.

Громкость воспроизведения же регулируется ползунком slVol, максимальное значение которого равно 100.

1. private void slTime\_Scroll(object sender, EventArgs e)
2. {
3. BassPlayer.SetPosOfScroll(BassPlayer.\_stream, slTime.Value);
4. }
5. BassPlayer.SetVolumeToStream(BassPlayer.\_stream, slVol.Value);

## **3.3 Интеграция eq.cs**

Для интеграции класса эквалайзера eq.cs не обязательно применять класс BassPlayer.cs. Это может быть любой другой плеер, написанный под bass.dll.

Теоретически возможно соединить аудио проигрыватель, написанный на одном языке, и эквалайзер на другом. При условии, что оба элемента написаны под библиотекой bass.dll

Для работы эквалайзер требует только аудио поток, на который будет применяться эквализация.

Главное преимущество данного эквалайзера, это возможность создать любое количество эквализационных полос. Структура модуля устроена таким образом, что в ней не указано строгое количество частотных полос, управление ими осуществляется всего тремя массивами данных.

1. public static int[] fx = new int[] { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 };
2. public static int[] freq = new int[] { 50, 100, 250, 500, 1000, 2500, 5000, 8000, 12000, 15000 };
3. public static int[] eqValues = new int[] { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 };

И именно в этих массивах и устанавливается количество эквализационных полос. Единственное что от них требуется, это последовательное соответствие значений друг другу. Это подразумевает, чтобы элемент каждого массива под одним индексом являли собой часть одной полосы. К примеру третий индекс массива fx должен быть неотрывно связан с частотой freq. А также ползунок частоты связанный с данной частотой, в функции изменения значений, должен ссылаться на третий элемент массива.

Только при корректной установке этих значений, эквалайзер будет корректно выполнять свои функции.

От количества полос зависит и добротность диапазона. Потому можно использовать любое количество полос, не беспокоясь что редактируемые частоты будут конфликтовать друг с другом.

Для эквалайзера необходимо создать управляющие элементы полос. Лучше всего для этого подходят ползунки значений. Наиболее удобным будет устанавливать значение в два раза выше пиковой громкости. Интерфейс эквалайзера представлен на рисунке 9.

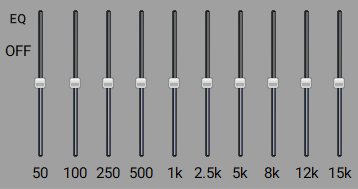


Рис.9. Интерфейс эквалайзера.

Если максимальная громкость равна 10, то максимальное значение устанавливается на 20, и нейтральным положением становится также 10. И каждый раз при изменении значения из текущего положения отнимается также 10.

Такой алгоритм позволяет из ползунков получать так же отрицательные значения. Ведь помимо увеличения громкости, эквалайзеры так же ее уменьшают. И таким способом можно установить диапазон отклонения амплитуды равный по 10 дБ в каждую сторону.

Для активации эквалайзера помимо массивов требуется создать еще одну переменную. Логическая переменная EqIsActive будет хранить состояние эквалайзера.

После нажатия кнопки активации, сперва происходит проверка, был ли эквалайзер уже запущен, и если нет, то производит его инициализацию.

Вызывая функцию ActivEq речь о которой шла в разделе 2.4, в нее передаются те три основных массива о которых ранее шла речь, и они автоматически инициализируются, подключаясь к аудио потоку.

В этот момент переменная состояния эквалайзера изменяя свое значение. И это позволяет использовать одну кнопку для включения и отключения эквалайзера.

Для визуального определения состояния эквалайзера, можно использовать различные элементы. В данном случает используется текстовый контейнер, расположенный прямо под кнопкой эквалайзера.

1. private void eqButton\_Click(object sender, EventArgs e)
2. {
3. if (!Main.EqIsActive)
4. {
5. eq.ActivEq(Main.eqValues, Main.fx, Main.freq, BassPlayer.\_stream);
6. Main.EqIsActive = true;
7. eqTextSwitch.Text = "ON";
8. } else
9. {
10. eq.DeactivEq(Main.fx);
11. Main.EqIsActive = false;
12. eqTextSwitch.Text = "OFF";
13. }
14. }

Так как эквалайзер это устройство работающее в реально времени, то при активации эквалайзера, все изменения должны применяться незамедлительно. Для этого на каждый ползунок эквализации, создается функция срабатывающая при изменении ее значения.

Функция Scroll позволяет выполнять определенные действия, сразу после того, как значение ползунка было изменено.

Рассмотрим это на примере ползунка частоты 50 Гц.

Ползунок имеет специальное название sl50hz, которое обозначает что этот ползунок отвечает за частоту в 50 герц.

Первым действием новое значение присваивается в соответствующую ячейку значений громкости. Предварительное вычитание от значения 10 единиц, позволяет держать все значения в диапазоне от – 10 до 10 дБ.

Затем после проверки состояния эквалайзера, вызывается функция UpdateEq() которая принимает все три основных массива, а также индекс полосы эквализации. Например для ползунка sl250hz индекс будет равняться 2.

private void sl50hz\_Scroll(object sender, EventArgs e)

{

Main.eqValues[0] = sl50hz.Value - 10;

if (Main.EqIsActive == true)

eq.UpdateEq(Main.eqValues, Main.fx, Main.freq, 0);

}

Первостепенная запись нового значения в массив, позволяет даже при выключенном эквалайзере заранее подготовить необходимые частоты перед его активацией.

Весь код представлен в приложении 1.

Если бы все значения передавались напрямую от ползунков, первостепенная активация эквалайзера стала бы невозможной, или по крайней мере заняло очень большой объем кода. Потому использование массива и его обновление при каждом изменении значения ползунка является обязательным.

# Заключение

В результате выполнения выпускной квалификационной работы были изучены основы разработки программных модулей. Был разработан интегрируемый программный модуль эквалайзера под библиотекой bass.dll. В выпускной квалификационной работе были проанализированы процессы, сопровождающие создание компьютерного приложения. Был описан процесс внедрения программных модулей в код, не только эквалайзера, но и аудиоплеера.

Программный модуль готов, и в любое время может быть использован разработчика программного обеспечения, для улучшения сервиса своих приложений.

Таким образом, в результате выполнения выпускной квалификационной работы, были выполнены все поставленные задачи. Разработанный программный продукт соответствует всем нормам и требованиям, которые были заявлены.

# Список литературы

1. Теория звука. Что нужно знать о звуке, чтобы с ним работать. Опыт Яндекс.Музыки.
2. Айфичер Э., Джервис Б. — Цифровая обработка сигналов. Практический подход.
3. Вахитов Я.Ш., Смирнонва Н.А. и др. «Акустика и электроакустика. Теоретические основы электроакустики и психофизика слуха»
4. Дворко Н.И. «Звукорежиссура»
5. Дункан Фрай «Микширование живого звука»
6. Ершов К.Г., Беспрозванный М.В «Оборудование звукотехнических комплексов киностудий»
7. Загуменов А.П. «Запись и редактирование звука. Музыкальные эффекты»
8. Иванова, Г. С. Объектно-ориентированное программирование
9. Клюкин И.И. «Удивительный Мир звука»
10. Курбат К. «Звукооператор-любитель»
11. Леонтьев В.П. «Обработка музыки и звука на компьютере. Быстро и качественно»
12. Маньковский В.С. «Основы звукооператорской работы»
13. Никамин В.А. «Цифровая звукозапись. Технология и стандарты»
14. Нисбетт Алекс «Звуковая студия. Техника и методы использования»
15. Стаценко Л.Г. Паскаль Ю.В. «Акустика студий звукового и телевизионного вещания. Системы озвучивания»
16. Федорова Г.Н Разработка программных модулей программного обеспечения для компьютерных систем

# Приложение

Программный код класса MainWindow.cs

private void importButton\_Click(object sender, EventArgs e)

{

openFileDialog1.ShowDialog();

}

/// <summary>

/// запись выбраных для воспроизведения файлов в массив проигрываемых файлов

/// </summary>

/// <param name="sender"></param>

/// <param name="e"></param>

private void openFileDialog1\_FileOk(object sender, CancelEventArgs e)

{

Main.\_files.Add(openFileDialog1.FileName);

playList.Items.Add(Main.GetFileName(openFileDialog1.FileName));

}

/// <summary>

/// активация воспроизведения кнопкой P

/// </summary>

/// <param name="sender"></param>

/// <param name="e"></param>

private void playButton\_Click(object sender, EventArgs e)

{

if((playList.Items.Count != 0) && (playList.SelectedIndex != -1))

{

string current = Main.\_files[playList.SelectedIndex];

BassPlayer.Play(current);

trackTime.Text = TimeSpan.FromSeconds(BassPlayer.GetTimeOfStream(BassPlayer.\_stream)).ToString();

slTime.Maximum = BassPlayer.GetTimeOfStream(BassPlayer.\_stream);

timer.Enabled = true;

}

}

/// <summary>

/// Остановка воспроизведения по кнопке S

/// </summary>

/// <param name="sender"></param>

/// <param name="e"></param>

private void stopButton\_Click(object sender, EventArgs e)

{

BassPlayer.Stop();

timer.Enabled = false;

slTime.Value = 0;

playTime.Text = "00:00:00";

}

/// <summary>

/// отсчет таймера для обновления показателей

/// </summary>

/// <param name="sender"></param>

/// <param name="e"></param>

private void timer\_Tick(object sender, EventArgs e)

{

playTime.Text = TimeSpan.FromSeconds(BassPlayer.GetPosOfStream(BassPlayer.\_stream)).ToString();

slTime.Value = BassPlayer.GetPosOfStream(BassPlayer.\_stream);

}

/// <summary>

/// функция перемотки трека посредством слайдера прогресса проигрывания

/// </summary>

/// <param name="sender"></param>

/// <param name="e"></param>

private void slTime\_Scroll(object sender, EventArgs e)

{

BassPlayer.SetPosOfScroll(BassPlayer.\_stream, slTime.Value);

}

/// <summary>

/// функция изменения звука посредством слайзера звука

/// </summary>

/// <param name="sender"></param>

/// <param name="e"></param>

private void slVol\_Scroll(object sender, EventArgs e)

{

BassPlayer.SetVolumeToStream(BassPlayer.\_stream, slVol.Value);

}

/// <summary>

/// Приостановка проигрывания кнопкой U

/// </summary>

/// <param name="sender"></param>

/// <param name="e"></param>

private void pauseButton\_Click(object sender, EventArgs e)

{

BassPlayer.Pause();

timer.Enabled = false;

}

/// <summary>

/// обновление данных эквалайзера при измении значения

/// </summary>

/// <param name="sender"></param>

/// <param name="e"></param>

private void sl50hz\_Scroll(object sender, EventArgs e)

{

Main.eqValues[0] = sl50hz.Value - 10;

if (Main.EqIsActive == true)

eq.UpdateEq(Main.eqValues, Main.fx, Main.freq, 0);

}

private void sl100hz\_Scroll(object sender, EventArgs e)

{

Main.eqValues[1] = sl100hz.Value - 10;

if (Main.EqIsActive == true)

eq.UpdateEq(Main.eqValues, Main.fx, Main.freq ,1);

}

private void sl250hz\_Scroll(object sender, EventArgs e)

{

Main.eqValues[2] = sl250hz.Value - 10;

if (Main.EqIsActive == true)

eq.UpdateEq(Main.eqValues, Main.fx, Main.freq, 2);

}

private void sl500hz\_Scroll(object sender, EventArgs e)

{

Main.eqValues[3] = sl500hz.Value - 10;

if (Main.EqIsActive == true)

eq.UpdateEq(Main.eqValues, Main.fx, Main.freq, 3);

}

private void sl1khz\_Scroll(object sender, EventArgs e)

{

Main.eqValues[4] = sl1khz.Value - 10;

if (Main.EqIsActive == true)

eq.UpdateEq(Main.eqValues, Main.fx, Main.freq, 4);

}

private void sl2\_5khz\_Scroll(object sender, EventArgs e)

{

Main.eqValues[5] = sl2\_5khz.Value - 10;

if (Main.EqIsActive == true)

eq.UpdateEq(Main.eqValues, Main.fx, Main.freq, 5);

}

private void sl5khz\_Scroll(object sender, EventArgs e)

{

Main.eqValues[6] = sl5khz.Value - 10;

if (Main.EqIsActive == true)

eq.UpdateEq(Main.eqValues, Main.fx, Main.freq, 6);

}

private void sl8khz\_Scroll(object sender, EventArgs e)

{

Main.eqValues[7] = sl8khz.Value - 10;

if (Main.EqIsActive == true)

eq.UpdateEq(Main.eqValues, Main.fx, Main.freq, 7);

}

private void sl12khz\_Scroll(object sender, EventArgs e)

{

Main.eqValues[8] = sl12khz.Value - 10;

if (Main.EqIsActive == true)

eq.UpdateEq(Main.eqValues, Main.fx, Main.freq, 8);

}

private void sl15khz\_Scroll(object sender, EventArgs e)

{

Main.eqValues[9] = sl15khz.Value - 10;

if (Main.EqIsActive == true)

eq.UpdateEq(Main.eqValues, Main.fx, Main.freq, 9);

}

/// <summary>

/// активация эквалайзера кнопкой EQ

/// </summary>

/// <param name="sender"></param>

/// <param name="e"></param>

private void eqButton\_Click(object sender, EventArgs e)

{

if (!Main.EqIsActive)

{

eq.ActivEq(Main.eqValues, Main.fx, Main.freq, BassPlayer.\_stream);

Main.EqIsActive = true;

eqTextSwitch.Text = "ON";

} else

{

eq.DeactivEq(Main.fx);

Main.EqIsActive = false;

eqTextSwitch.Text = "OFF";

}

}

}

}