

BeatSync

מגיש: וולדיסלב פוברז'ני

כיתה: י"ג הנדסת תוכנה

בית ספר: מכללת הכפר הירוק

מנחה: יהודה אור

שפות תיכנות: C#, XML, Arduino

סביבות פיתוח: Visual Studio IDE (Xamarin), Arduino IDE

מבוא

פרויקט זה נועד להעשיר את החוויה של ספורטאים והעוסקים בפעילות גופנית סדירה.

העשרת החוויה נעשית על ידי התאמת קצב המוזיקה, אותה שומע הספורטאי במכשיר הטלפון הנייד שלו לקצב פעימות הלב המדדות בו זמנית. כמו כן מוצגות על מסך הטלפון קצב פעימות הלב והמוסיקה וגם שם השיר הנבחר אוטומטית מתוך רשימה נבחרת.

מטרת הפרויקט היא להראות יכולת שליטה בפיתוח תוכנה במרחב רחב דיסציפלינות הכולל חומרה ותוכנה כמפורט:

- יסודות עיבוד אותות דיגיטליים (DSP – Digital Signal Processing)
- יישום של פונקציות מתמטיקה לעיבוד אותות דיגיטליים – כדוגמת FFT
- עבודה עם המבנה של הפורמט WAVE (כתיבת מפענח) לצורך העיבוד של אודיו
- עבודה בתחום בקרי תכנות Arduino
- בניית מעגלים חשמליים לחיבור כל מרכיבי החומרה של הפרויקט
- יישום פרוטוקול תקשורת Bluetooth לצורך העברת מידע
- יצירת יישום עבור הטלפון החכם על פלטפורמת אנדרואיד הכולל גרפיקה
- מימוש ידע זה באמצעות תכנות (קוד)
- תכנות בשפות C#, XML, Arduino

תוכן עניינים

1.....	<u>רפלקציה</u>
2.....	<u>תאור טכני</u>
3.....	<u>אופן הביצוע ורקע תיוארטי</u>
3.....	חישן דופק
3.....	Bluetooth רכיב
4.....	אפליקציה BeatSync
5.....	פורמט WAVE
7.....	DFT vs FFT – התמרת פורייה
12.....	<u>נספחים</u>
12.....	קוד הפרוייקט
69.....	מפרטי חומרה
70.....	מילון מונחים
71.....	מקורות

רפלקציה

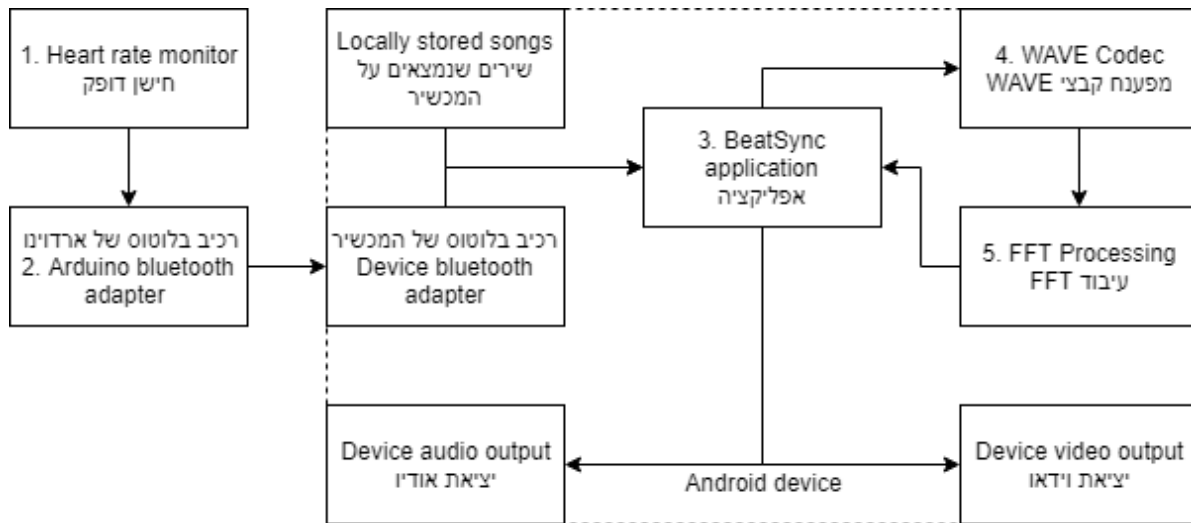
נהנתי מאוד לעבוד על הפרויקט שלי. התחום של מוזיקה והתצוגה שלה באופן דיגיטלי תמיד ענייני וסקרנו אותי. פרויקט זה נתן לי אפשרות לחקור איך התהליך של דיגיטיזציה של שירים שאנו שומעים כל יום עובדת מאחורי הקלעים. יתר על כן, חקירה בתחומים האלו לימדה אותי לעבד את אינפורמציה הדיגיטלית ולחשב מידע נוסף שימושי.

בחרתי את הנושא המסוים הזה לא רק בגלל הסקרנות האישית שלי, אלא גם בגלל שהרעיון של הפרוייקט בולט מפרויקטים רגילים. קיבלתי אפשרות לייצר משהו שלא רק עוזר לי ללמוד ולהתקדם, אבל גם מניע אחרים להצליח במטרות הספורטיביות שלהם.

בזמן של כתיבה וחקירה על הפרויקט יצא לי ללמוד על מימוניות שחדשות לי:

- חקרתי על פורמטים RIFF ו-WAVE, ועל המבנה הבינארי שלהם, איך להוציא מידע שימושי מקבצים האלו לשנות וגם לכתוב מידע זו מחדש לעיבוד על ידי תוכנות אודיו אחרות.
- למדתי לעבד אותות דיגיטליים של אודיו ולהשתמש בנוסחאות מתמטיקה מתקדמות כדי להעביר אותות מדומיין של זמן לדומיין של תדירויות ובחזרה.
- חקרתי בתחום של בקרי ארדוינו, חישנים ומעגלים חשמליים.
- למדתי על עבודה עם אפליקציות על פלטפורמת אנדרואיד ובניה של אפליקציות משלי.
- חקרתי על שידור אלחוטי של מידע דרך בלוטוס, עבודה עם סוקטים והחלפת אינפורמציה בין שרתים ולקוחות.
- פיתחתי אוטוידקטיות – יכולת למידה עצמית ממקורות מידע שונים (כמו אינטרנט).

תאור טכני



תרשים 1

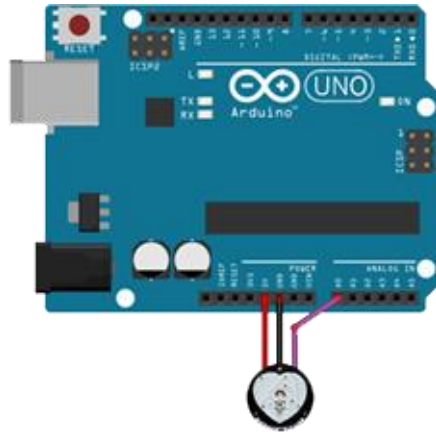
מבחינה טכנית, פרויקט זה הוא מכשיר קטן המכיל חישן דופק מחובר לשלט ארדואינו שמתקשר דרך רכיב בלוטוס עם אפליקציה בטלפון של המשתמש (תרשים 1).

מכשיר זה מונח על היד ומוודד את הדופק של אדם במהלך פעילות גופנית. תוצאות המדידות מוגשות ליישום בטלפון ומפורשות על המסך. אפליקציה BeatSync סורקת את כל הקבצי מוזיקה בפורמט WAV ומחשבת את הקצב של כל שיר הנמצא על הטלפון של המשתמש.

בהתאם לשינויים בקצב הלב, אפליקציה בוחרת שיר לנגינה עם קצב הכי מתאים לקצב פעימות הלב החדש והמוזיקה משתנה לפי הפלייליסט האישי של האדם המוגדר באפליקציה. לכן, המשתמש יכול לפקח על הדופק שלו, ולשמור אותו בתוך מגבלת הגיל (המוגדות גם בתוכנה) במהלך פעילות גופנית או ספורט, ובו זמנית ליהנות מהמוזיקה האהובה עליו.

אופן הביצוע ורקע תיאורטי

1. חישן דופק



תרשים 2

חישן דופק שנשתמש בו בפרוייקט הזה הוא SEN-11574 מחובר לבקר ארדואינו (תרשים 2).

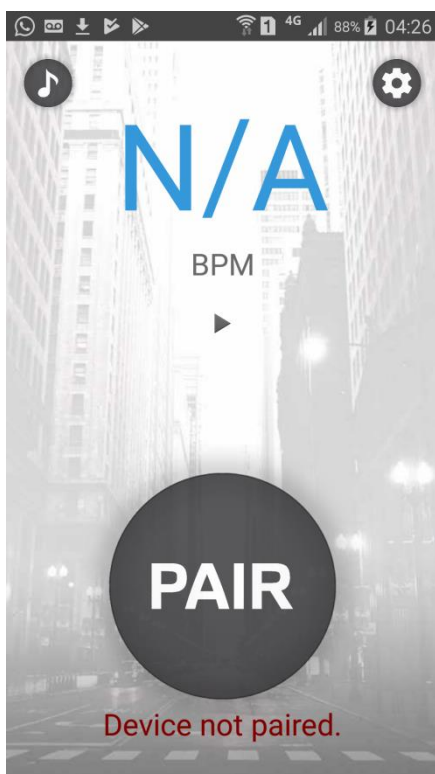
2. רכיב Bluetooth



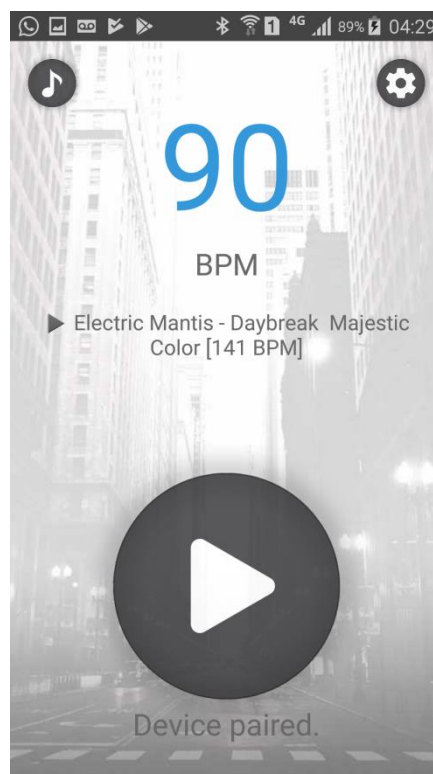
תרשים 3

רכיב בלוטוס המחובר למערכת שלנו הוא HC-05 (תרשים 3). חוץ מפינים GND (GROUND) ו-VCC (VOLT) אנו גם נחבר את TXD (TRANSMITTER) שיתן לרכיב אפשרות לשדר מידע.

3. אפליקציה BeatSync



תרשים 5



תרשים 4

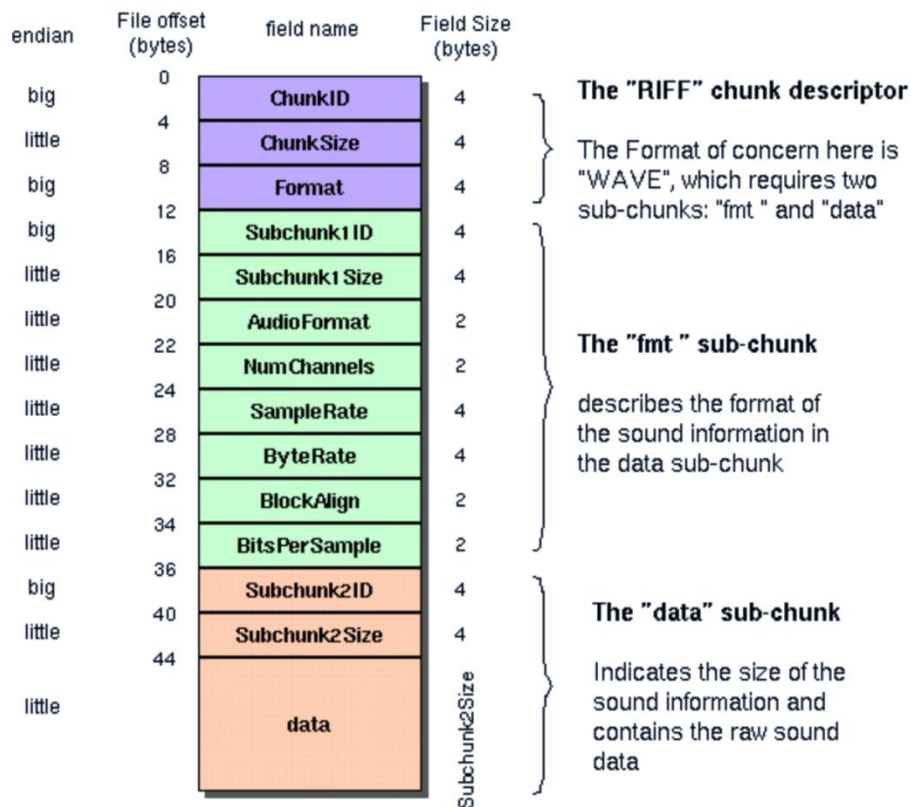
אפליקציה **BeatSync** הוא כלי שמחבר את כל החלקי הפרוייקט ביחד. אפליקציה זו כתובה למכשירי אנדרואיד עם רכיבי Bluetooth המסוגלים לתקשר עם רכיב HC-05 ולקבל מידע מרכיב זה. תוכנה זו גם אחראית על סריקת השירים בטלפון של המשתמש בפורמט WAVE, חישוב הקצב של כל שיר והתאמה של שיר המתנגן לקצב פעימות הלב של המשתמש בזמן פעילות גופנית.

לאפליקציה יש שני מצבים – Paired (תרשים 4) ו-Not Paired (תרשים 5). בשני המצבים המשתמש יכול לקבל מידע על הקצב של כל שיר בפורמט WAVE הנמצא על המכשיר שלו ולנגן אותם. אחרי הלחיצה על כפתור הנגינה במצב Paired (כשקיים קשר בין חישן הדופק והמכשיר של המשתמש) אפליקציה תשלוח בנגינה של השירים לפי מידע המתקבל מרכיב הבלוטוס של הארדוויין.

אפליקציה תפסיק את נגינת המוסיקה ותתריע על המעבר העליון של קצב פעימות הלב של המשתמש, טווח המוגדר ע"פ גיל המשתמש. אחרי החזרת הקצב לטווח המותר האפליקציה תחזור לנגן מוסיקה.

כשאפליקציה לא נמצאת במצב של נגינה, למשתמש יש שליטה רגילה והוא יכול נגן כל שיר נבחר – המשתמש יכול לבחור איזה שירים מותר לאפליקציה לנגן בזמן פעולות פיזיות (לבחור פלייליסט), לנגן שירים שעברו עיבוד על ידי המפענח, לעצור ולהמשיך נגינה של השיר בכל נקודת זמן.

The Canonical WAVE file format

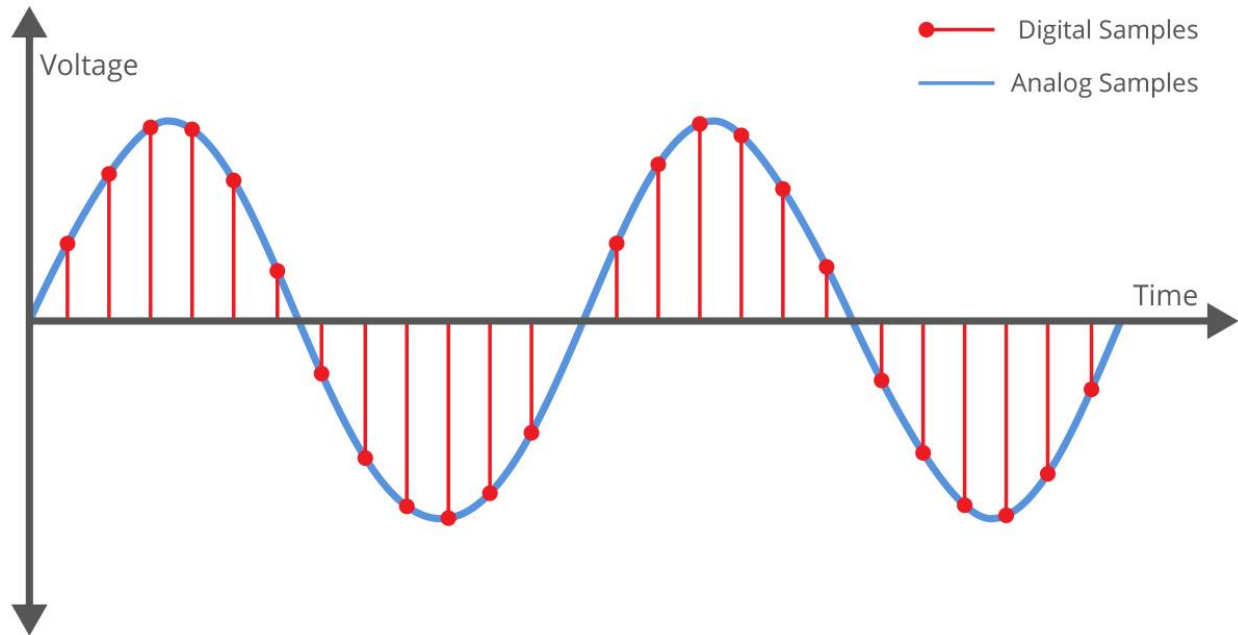


תרשים 6

פורמט WAVE הוא פורמט אודיו רשמי של מיקרוסופט מפותח באמצעות שיטה של שמירת מידע **RIFF** (Resource Interchange File Format). שיטה זו מחלקת מידע לכמה "נתחים" (chunks). בניגוד לפורמט AVI המפותח באמצעות RIFF המכיל בתוכו גם וידאו ואודיו, פורמט WAVE יכול להחזיק רק מידע קולית.

קבצים WAVE חייבים להכיל שלושת הנתחים הנתונים ב**תרשים 6**: "RIFF", "fmt", ו-"data". נתח "RIFF" מגדיר את הקובץ כתור קובץ RIFF, נותן מידע על גודל הקובץ ומגדיר סוג קובץ מסוג WAVE. נתח "fmt" מכיל מידע על נתונים גולמיים השמורים בקובץ. מידע זה מכיל אינפורמציה חשובה עבור השמעת אודיו כמו קצב דגימה (Sample Rate), קצב נתונים (Bit Rate) וכו'. נתח "data" שומר את כל הנתונים הגולמיים – מידע בכי חשוב בקובץ שנצטרך לעבד בהמשך.

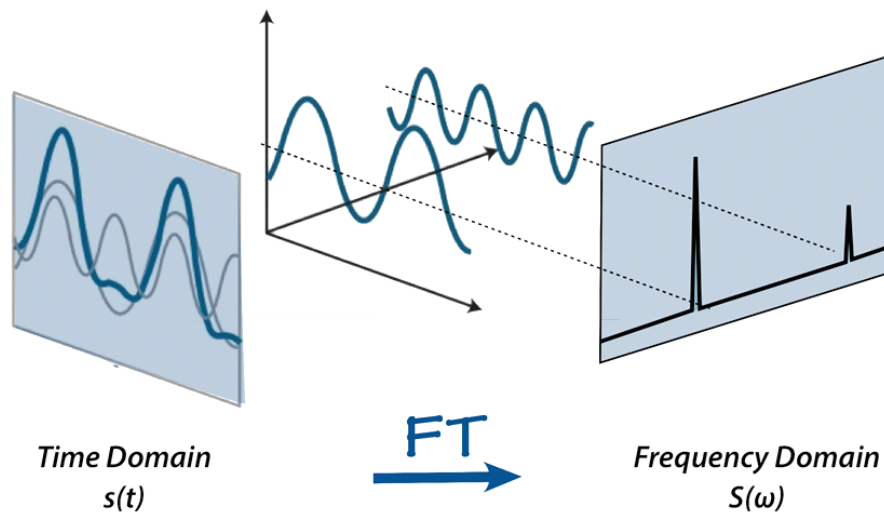
דגימה ו-DSP – עיבוד אותות דיגיטליים



תרשים 7

DSP (Digital Signal Processing) הוא תחום העוסק באותות דיגיטליים ועיבוד של אותות אלה. אחרי שאות אנלוגי עובר תהליך של דיגיטליזציה בשם **דגימה (Sampling)**, הוא מתחלק לדגימות שמחקות את הצורה של אות הנקלט (**תרשים 7**) ונהיה מוכן לעיבוד DSP. ניתן לעבד אותות אודיו דיגיטליים באמצעות כלים שונים – מיקסרים (מערבלים), אקולייזרים (משוים), פילטרים (מסננים) וכו'. לצורך החישוב הקצב של שירים נצטרך לבנות מסנן משלנו.

5. DFT vs FFT – התמרת פורייה



תרשים 8

FT (Fourier Transform) היא התמרה שמפרקת פונקציות של זמן (אותות) לתדרים שמרכיבים אותן. התמרת פורייה היא אחד הכלים החשובים בעיבוד אותות אודיו שמאפשר להעביר אות מתחום הזמן לתחום התדר (**תרשים 8**) ונותן לנו שליטה בתדירויות שמרכיבים את האות הנתון.

$$X(k) = \sum_{n=0}^{N-1} x(n) e^{-i2\pi kn/N}$$

תרשים 9

התמרת פורייה מתמשכת נקראת **DFT (Discrete Fourier Transform)** המאפשרת לעבד אות באורך N דגימות. את ההתמרה DFT ניתן לממש עם פונקציה מתמטית נתונה ב**תרשים 9**. חסרון של וריאציה הזו של התמרת פורייה הוא שסיבוכיות זמן ריצה של אלגוריתם DFT ממומש מקוד היא $O(n^2)$. משך זמן החישוב תלוי בכמות הדגימות וגודל בצורה אקספוננציאלית – חישוב DFT עבור שיר של 5 דקות יכול לקחת שעות.

Sample numbers
in normal order

Decimal Binary

0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001
10	1010
11	1011
12	1100
13	1101
14	1110
15	1111

Sample numbers
after bit reversal

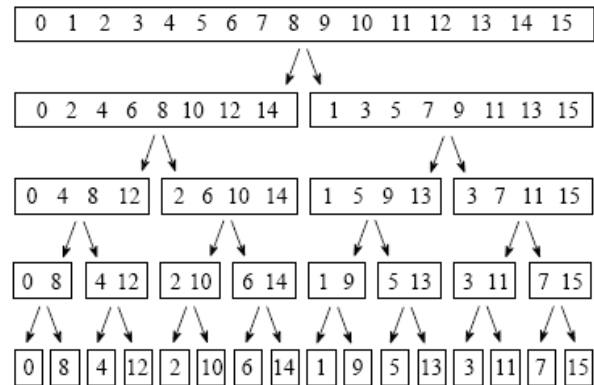
Decimal Binary

0	0000
8	1000
4	0100
12	1100
2	0010
10	1010
6	0110
14	1110
1	0001
9	1001
5	0101
13	1101
3	0011
11	1011
7	0111
15	1111



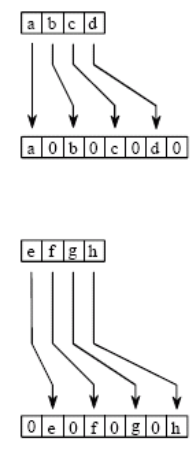
תרשים 11

1 signal of
16 points
2 signals of
8 points
4 signals of
4 points
8 signals of
2 points
16 signals of
1 point

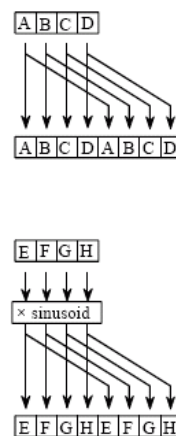


תרשים 10

Time Domain



Frequency Domain



תרשים 12

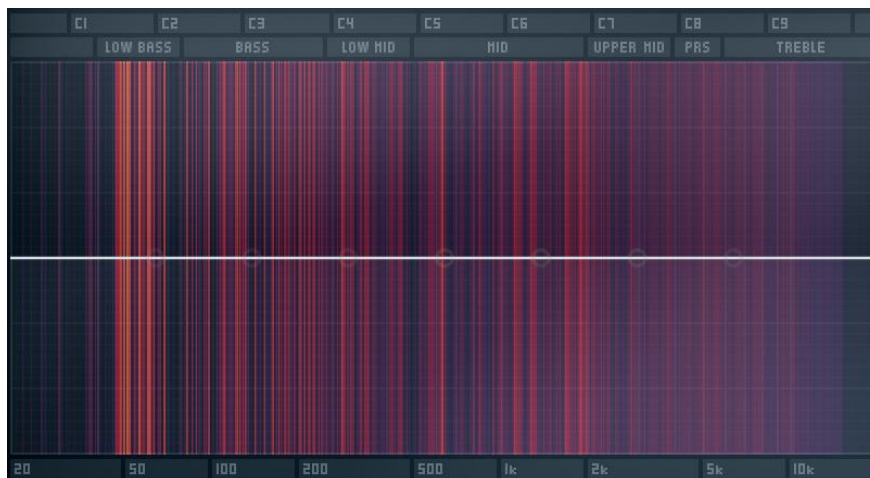
כדי לקטן את זמני החישוב שלנו, נשתמש בוריאציה אחרת של התמרת פורייה בשם **FFT (Fast Fourier Transform)**. אלגוריתם FFT עובד באופן הבא:

- (1) חלוקה של כל האות לאותות קטנים יותר (**תרשים 10**)
- (2) מיון של חלקים החדשים עם אלגוריתם Bit Reversal (**תרשים 11**)
- (3) חישוב DFT עבור החלקים האלו בנפרד
- (4) שילוב של ספקטרום באופן בדיוק הפוך מאופן שבו פרקנו את האות בתחום הזמן (**תרשים 12**)

סיבוכיות זמן ריצה של אלגוריתם FFT היא $O(n \cdot \log(n))$. נשים את זה בפרספקטיבה – עבור חישוב של 1024 דגימות, אלגוריתם FFT הוא פי 102.4 יותר מהר מ-DFT. שיפור ביעילות אלגוריתם מאפשר לנו לחשב תחום התדירויות עבור שירים שלמים תוך שניות! למרות התוצאות משופרות שקיבלנו, התמרה FFT היא לא מושלמת – שימוש באלגוריתם זה מגביל את כמות הדגימות הזמינות לחישוב

לחזקה של 2. זאת אומרת שאנו מסוגלים לעבד אותות אך ורק באורך של 2^h דגימות. לדוגמה, רק אותות באורך של 2^8 256, 2^{10} 1024, 2^{16} 65536 וכו' דגימות נתונים לחישוב על ידי התמרה FFT. קיימות שיטות שונות שעוקפות את ההגבלה זו (כמו ריפוד אות עם אפסים עד החזקה הקרובה ביותר של 2), אך כל שיטה פוגעת ביעילות וזמן עיבוד של אלגוריטם. לכן, כדי לחשב תחום התדר מהר יותר, נצטרך לעבד אותות קטנים יותר.

כתיבת מסנן אודיו



תרשים 13



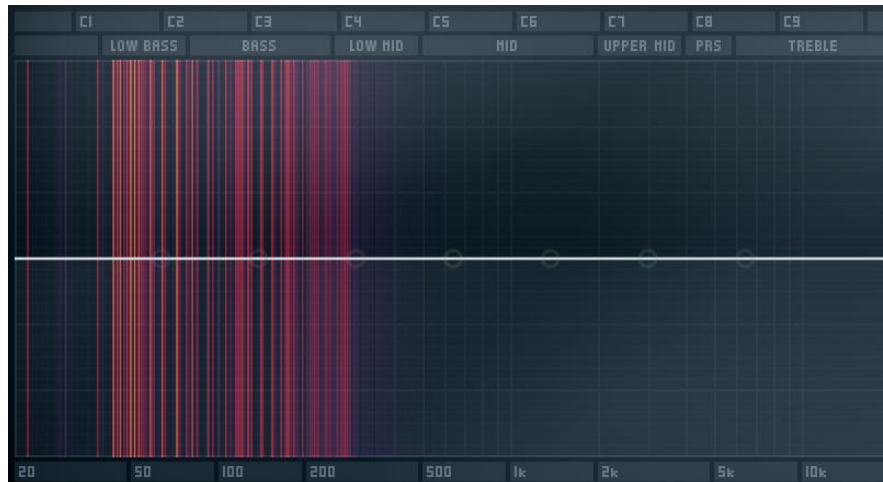
תרשים 14

מסנן אודיו (Audio Filter) הוא כלי עיבוד אותות דיגיטליים בסיסי. תפקידו הראשי של מסנן אודיו הוא לקבל אות ולסנן טווח תדירויות מסוים המוגדר בהגדרות של המסנן, אף כי מסננים גם מסוגלים להגביר ולהחליש תדירויות המתקבלות מהאות. רוב המסנני אודיו מיקצעים עובדים עם טווח של תדירויות מוגבל בין 0 – 22000 הרץ בגלל שהטווח תדרים של שמיעה אנושית (טווח שהאוזניים שלנו מסוגלים לקלוט) הוא בין 20 – 20000 הרץ.

בגלל שמסנני אודיו עובדים עם תחום התדרים, הם חייבים לעבד אותות ולהעביר אותם מתחום הזמן. קיימות פונקציות שמסוגלות לבצע תפקידים בסיסיים של המסנן בתחום הזמן, אך רוב המסננים משתמשים בהתמרת פורייה מכיון שאלגוריתם זה נותן למשתמשים מגוון רחב יותר של פונקציונליות.

בתרשים 13 אפשר לראות את הטווח תדירויות של שיר מתנגן. תרשים זה הוא גרף של תדירות כפונקציה של עוצמה – תחום התדר של השיר. **בתרשים 14** אפשר לראות את צורת הגל של השיר. תרשים זה מציג את הזמן כפונקציה של עוצמה – תחום הזמן של השיר.

תופים הם אלמנט מוסיקלי בשיר שמגדיר את קצב של השיר. טווח תדרים של תופים הוא בין 20 – 210 הרץ. אם אנו לבודד את התדרים האלה עם מסנן אודיו, אנו נבטל את האלמנטים אחרים חסרי תועלת עבורנו ונוכל להתרכז רק בתופים.



תרשים 15



תרשים 16

תרשים 15 מראה לנו את התחום התדרים של השיר אחרי התהליך של סינון תדירויות מאל 250 הרץ. **תרשים 16** מראה לנו את צורת הגל של השיר אחרי סינון. כל פסגה שאנו רואים בתרשים זה הוא תוף המתנגן בשיר. אם נמצא בגרף הזה תוף שחוזר על עצמו לאחר אותה כמות זמן, אפשר לחשב לפי ההבדלים בזמן של התוף את הקצב שלו בפורמט BPM ולפי נתון הזה לחשב את הקצב של השיר עצמו.

נספחים

1. קוד הפרוייקט – Arduino IDE

```
1. // Set-up low-level interrupts for most accurate BPM math.
2. #define USE_ARDUINO_INTERRUPTS true
3. #include <PulseSensorPlayground.h>
4.
5. // Define variables
6. PulseSensorPlayground pulseSensor;
7. int nPulsePin = A0;
8. int nLEDPin = 13;
9. int nThreshold = 675;
10.
11. //-----
12. //                               BeatSync
13. //                               -----
14. //
15. // General : This code is a part of the BeatSync project and is written
16. //           for Arduino Nano with a bluetooth adapter and a heartbeat sensor.
17. //
18. // Input   : User lays their finger on the heartbeat sensor. Sensor then
19. //           calculates the user's BPM based on the difference between the
20. //           impulses that go above the given threshold.
21. //
22. // Output  : Program takes the calculated BPM and transmits it as a byte
23. //           over bluetooth.
24. //
25. //-----
26. // Programmer : Vlad Poberezhny
27. // Date : 03.04.2018
28. //-----
29. void setup()
30. {
31.   // Begin serial at 9600 baud
32.   // Bluetooth adapter can only transmit at 9600 baud or lower
33.   Serial.begin(9600);
34.
35.   // Set-up the pulse sensor
36.   pulseSensor.analogInput(nPulsePin);
37.   pulseSensor.blinkOnPulse(nLEDPin);
38.   pulseSensor.setThreshold(nThreshold);
39.
40.   // Begin reading
41.   pulseSensor.begin();
42. }
43.
44. void loop()
45. {
46.   // If heartbeat was detected
47.   if (pulseSensor.sawStartOfBeat())
48.   {
49.     // Convert to BPM, transmit calculation as byte
50.     Serial.write(pulseSensor.getBeatsPerMinute());
51.   }
52.
53.   // Small delay is a good practice
54.   delay(10);
55. }
```

קוד הפרוייקט – BeatSync – MainActivity.cs

```
1. using System.Linq;
2. using System.Collections.Generic;
3. using System.IO;
4. using Android.App;
5. using Android.Widget;
6. using Android.OS;
7. using Android.Views;
8. using Android.Bluetooth;
9. using Android.Content;
10. using Android.Graphics;
11. using Android.Support.V4.App;
12. using Android.Content.Res;
13.
14. namespace BeatSync
15. {
16.     [Activity(Label = "BeatSync",
17.               MainLauncher = true,
18.               ConfigurationChanges = Android.Content.PM.ConfigChanges.Orientation |
19.               Android.Content.PM.ConfigChanges.ScreenSize)]
20.
21.     public partial class MainActivity : Activity
22.     {
23.         // Current activity views
24.         private static TextView txtBPM;
25.         private static TextView txtStatus;
26.         private static ImageView btnMain;
27.         private static ImageView btnPlaylist;
28.         private static ImageView btnSettings;
29.         private static TextView txtSong;
30.
31.         // Adapters and receivers
32.         private static BluetoothAdapter PhoneAdapter;
33.         private static ArduinoBroadcastReceiver Receiver;
34.
35.         // User related variables
36.         private static int userAge;
37.         private static int userMinBPM;
38.         private static int userMaxBPM;
39.
40.         // Other variables
41.         private static float btnMain_DefaultAlpha;
42.
43.         /// <summary>
44.         /// Changes the text and color of the status label.
45.         /// </summary>
46.         /// <param name="status">New status text.</param>
47.         /// <param name="color">Color of the new text.</param>
48.         protected static void SetStatus(string status,
49.                                         Color color)
50.         {
51.             // If status label is initialized
52.             if (txtStatus != null)
53.             {
54.                 // Change status text and color
55.                 txtStatus.Text = status;
56.                 txtStatus.SetTextColor(color);
57.             }
58.         }
59.     }
60. }
```



```

59.
60.     /// <summary>
61.     /// Sets the current age of user.
62.     /// Calculates and returns the safe BPM range for user.
63.     /// </summary>
64.     /// <param name="age">User age.</param>
65.     /// <returns>Safe BPM range.</returns>
66.     public static int[] SetAge(int age)
67.     {
68.         userAge = age;
69.         userMaxBPM = (int)((BEATSYNC_CONSTANTS.BEATSYNC_MAXBPM -
userAge) * 0.85);
70.         userMinBPM = (int)((BEATSYNC_CONSTANTS.BEATSYNC_MAXBPM -
userAge) * 0.40);
71.         return (new int[] { userMinBPM, userMaxBPM });
72.     }
73.
74.     /// <summary>
75.     /// Changes text of the song label.
76.     /// </summary>
77.     /// <param name="label">Song title.</param>
78.     public static void SetSongLabel(string label)
79.     {
80.         txtSong.Text = "> " + label; // ■ ■ ▶ ■ ■ ■ ■ ■
81.     }
82.
83.     /// <summary>
84.     /// Function returns the key of the given value in the given dictionary.
85.     /// </summary>
86.     /// <param name="dictionary">Given dictionary.</param>
87.     /// <param name="value">Given dictionary value.</param>
88.     /// <returns>Dictionary key by value.</returns>
89.     public static string GetKeyByValue(Dictionary<string, string> dictionary,
90.                                         string value)
91.     {
92.         return (dictionary.FirstOrDefault(item => item.Value.Equals(value)).Key);
93.     }
94.
95.     /// <summary>
96.     /// Called when result for a request has been received.
97.     /// </summary>
98.     /// <param name="requestCode">Code of the request.</param>
99.     /// <param name="resultCode">Result of the request.</param>
100.    /// <param name="data">Data received.</param>
101.    protected override void OnActivityResult(int requestCode,
102.                                              Result resultCode,
103.                                              Intent data)
104.    {
105.        // Set main button alpha back to default
106.        btnMain.Alpha = btnMain_DefaultAlpha;
107.
108.        // If received result code is "ENABLE BLUETOOTH" code AND result is
"OK"
109.        if (requestCode == BEATSYNC_CONSTANTS.REQUEST_ENABLE_BLUETOOTH &&
110.            resultCode == Result.Ok)
111.        {
112.            // Change status
113.            SetStatus("Searching...", BEATSYNC_CONSTANTS.Colors.TextGray);
114.
115.            // Cancel discovery if device is already discovering
116.            if (PhoneAdapter.IsDiscovering)

```

```

117.         {
118.             PhoneAdapter.CancelDiscovery();
119.         }
120.
121.         // Start discovering
122.         new BluetoothDiscoverTask().Execute(PhoneAdapter);
123.     }
124.
125.     // Otherwise no changes, make button clickable
126.     else
127.     {
128.         btnMain.Clickable = true;
129.     }
130. }
131.
132.     /// <summary>
133.     /// Sends a basic notification to user.
134.     /// </summary>
135.     /// <param name="context">Current application environment context.</para
136.     m>
137.     /// <param name="contentText">Notification text.</param>
138.     /// <param name="iconID">Notification icon ID.</param>
139.     protected static void SendNotification(string contentText,
140.                                           int iconID)
141.     {
142.         // Set-up notification
143.         NotificationCompat.Builder newNotification = new NotificationCompat.
144.         Builder(Application.Context);
145.         newNotification.SetPriority((int)NotificationPriority.Max);
146.         newNotification.SetDefaults((int)NotificationDefaults.All);
147.         newNotification.SetContentTitle(BEATSYNC_CONSTANTS.BEATSYNC_APPLICAT
148.         ION_NAME);
149.         newNotification.SetContentText(contentText);
150.         newNotification.SetSmallIcon(iconID);
151.
152.         // Display notification
153.         ((NotificationManager)Application.Context.GetService(Notificat
154.         ionService)).Notify(BEATSYNC_CONSTANTS.BEATSYNC_NOTIFICATION_ID,
155.         newNotification.Build());
156.     }
157.
158.     /// <summary>
159.     /// Receives a BPM value. Returns a path to song with the closest BPM to
160.     received value.
161.     /// </summary>
162.     /// <param name="BPM"></param>
163.     /// <returns></returns>
164.     protected static string PickSong(int BPM)
165.     {
166.         // Define KVP variable for the answer
167.         KeyValuePair<string, int> ClosestSong = new KeyValuePair<string, int>
168.         >(null, 255);
169.
170.         // Iterate over each analyzed song
171.         foreach (KeyValuePair<string, int> Song in BEATSYNC_GLOBALS.songBPMs
172.         )
173.         {
174.             if (new Java.IO.File(Song.Key).Exists() && // I
175.             f current song exists

```

```

168.             BEATSYNC_GLOBALS.userPlaylist.Contains(Song.Key) &&      // I
169.             f current song is in the playlist
169.             !Song.Key.Equals(BEATSYNC_GLOBALS.currentlyPlaying) && // I
170.             f it's not the currently playing song
170.             System.Math.Abs(Song.Value -
171.             BPM) < ClosestSong.Value) // If BPM difference is smaller than the current answer's
171.             {
172.                 // Set the current song to be the current answer
173.                 ClosestSong = new KeyValuePair<string, int>(Song.Key,
174.                 System.Math.Abs(
175.                 Song.Value - BPM));
175.             }
176.         }
177.
178.         // Return the path to the song with the closest BPM
179.         return (ClosestSong.Key);
180.     }
181.
182.     /// <summary>
183.     /// Special kind of receiver which only looks for the specific Arduino b
184.     /// </summary>
185.     protected class ArduinoBroadcastReceiver : BroadcastReceiver
186.     {
187.         public override void OnReceive(Context context,
188.         Intent intent)
189.         {
190.             // If a device was found through discovery
191.             if (intent.Action == BluetoothDevice.ActionFound)
192.             {
193.                 // Retrieve device data from the intent
194.                 BluetoothDevice Device = (BluetoothDevice)intent.GetParcelableExtra(BluetoothDevice.ExtraDevice);
195.
196.                 // If found device is our Arduino adapter
197.                 if (Device.Address == BEATSYNC_CONSTANTS.ARDUINO_ADDRESS)
198.                 {
199.                     // Stop adapter discovery, start pairing process
200.                     PhoneAdapter.CancelDiscovery();
201.                     SetStatus("Pairing...", BEATSYNC_CONSTANTS.Colors.TextGr
202.                     ay);
203.
204.                     // Try pairing
205.                     try
206.                     {
207.                         // Get RFCOMM socket from the adapter
207.                         BluetoothSocket ArduinoSocket = Device.CreateRfcommS
208.                         ocketToServiceRecord(BEATSYNC_CONSTANTS.ARDUINO_UUID);
209.
210.                         // If device is not paired, display pairing PIN on p
211.                         airing dialog
211.                         if (!PhoneAdapter.BondedDevices.Contains(Device))
212.                         {
213.                             SendNotification("Pairing PIN: " + BEATSYNC_CONS
214.                             TANTS.ARDUINO_PAIRING_PIN, Resource.Drawable.icon_bluetooth);
215.                         }
216.
217.                         // Connect to device
218.                         ArduinoSocket.Connect();

```

```

218.                                     // If no exceptions up to this point -
    devices are connected
219.                                     BEATSYNC_GLOBALS.bPaired = true;
220.
221.                                     // BEGIN TASK THAT UPDATES TEXT LABEL WHILE CONNECTI
    ON IS STABLE
222.                                     new BPMTask().ExecuteOnExecutor(AsyncTask.ThreadPool
    Executor, ArduinoSocket);
223.                                     }
224.
225.                                     // If pairing failed
226.                                     catch (Java.IO.IOException)
227.                                     {
228.                                     SetStatus("Device not paired.", BEATSYNC_CONSTANTS.C
    olors.Red);
229.                                     BEATSYNC_GLOBALS.bPaired = false;
230.                                     }
231.                                     }
232.
233.                                     // Make main button clickable
234.                                     btnMain.Clickable = true;
235.                                     }
236.                                     }
237.                                     }
238.
239.                                     /// <summary>
240.                                     /// Task that discovers bluetooth adapters around the device.
241.                                     /// </summary>
242.                                     protected class BluetoothDiscoverTask : AsyncTask<BluetoothAdapter, int,
    BluetoothAdapter>
243.                                     {
244.                                     /// <summary>
245.                                     /// Set main button icon to "Pairing".
246.                                     /// </summary>
247.                                     protected override void OnPreExecute()
248.                                     {
249.                                     btnMain.SetImageResource(Resource.Drawable.icon_pairing01);
250.                                     }
251.
252.                                     protected override BluetoothAdapter RunInBackground(params Bluetooth
    Adapter[] @params)
253.                                     {
254.                                     // If device is currently discovering, cancel discovery
255.                                     if (@params[0].IsDiscovering)
256.                                     {
257.                                     @params[0].CancelDiscovery();
258.                                     }
259.
260.                                     // Begin discovering bluetooth adapters
261.                                     @params[0].StartDiscovery();
262.
263.                                     // Wait until the discovery boolean is true
264.                                     while (!@params[0].IsDiscovering) ;
265.
266.                                     // Make 12 loops 1 second each while adapter is enabled
267.                                     for (int nSecond = 0;
268.                                     nSecond < 12 &&
269.                                     @params[0].IsEnabled &&
270.                                     @params[0].IsDiscovering;
271.                                     nSecond++)
272.                                     {

```

```

273.                // Publish current second
274.                PublishProgress(nSecond);
275.
276.                // Sleep for 1 second
277.                System.Threading.Thread.Sleep(1000);
278.            }
279.
280.            // Return the adapter
281.            return @params[0];
282.        }
283.
284.        /// <summary>
285.        /// Animates the main button icon.
286.        /// </summary>
287.        /// <param name="values">Seconds passed since start of discovery.</p
aram>
288.        protected override void OnProgressUpdate(params int[] values)
289.        {
290.            // If amount of seconds is even, set to first frame
291.            if (values[0] % 2 == 0)
292.            {
293.                btnMain.SetImageResource(Resource.Drawable.icon_pairing01);
294.            }
295.
296.            // Otherwise set to second frame
297.            else
298.            {
299.                btnMain.SetImageResource(Resource.Drawable.icon_pairing02);
300.            }
301.        }
302.
303.        protected override void OnPostExecute(BluetoothAdapter Adapter)
304.        {
305.            // If adapter was disabled OR adapter is still discovering
306.            if (!Adapter.IsEnabled ||
307.                Adapter.IsDiscovering)
308.            {
309.                // Discovery was unsuccessful
310.                Adapter.CancelDiscovery();
311.                txtBPM.Text = "N/A";
312.                SetStatus("Device not paired.", BEATSYNC_CONSTANTS.Colors.Red);
313.                btnMain.SetImageResource(Resource.Drawable.txt_pair);
314.            }
315.
316.            // If devices are paired
317.            else if (BEATSYNC_GLOBALS.bPaired)
318.            {
319.                // Change program status
320.                SetStatus("Device paired.", BEATSYNC_CONSTANTS.Colors.TextGray);
321.                btnMain.SetImageResource(Resource.Drawable.icon_play);
322.            }
323.
324.            // Make main button clickable
325.            btnMain.Clickable = true;
326.        }
327.    }
328.

```

```

329.         /// <summary>
330.         /// Task receives BPM data over given stream and manages
331.         /// song playback based on BPM average.
332.         /// </summary>
333.         protected class BPMTask : AsyncTask<BluetoothSocket, int, bool>
334.         {
335.             // BPM related variables
336.             private int averageBPM      = 0;
337.             private int summatedBPM     = 0;
338.             private int updateCounter   = 0;
339.
340.             /// <summary>
341.             /// Receives the BPM data over the given stream.
342.             /// Passes each new BPM value to 'PublishProgress'.
343.             /// </summary>
344.             /// <param name="params">Data stream.</param>
345.             /// <returns></returns>
346.             protected override bool RunInBackground(params BluetoothSocket[] @pa
rams)
347.             {
348.                 // Set-up the adapter input stream
349.                 Stream ArduinoStream = @params[0].InputStream;
350.
351.                 // Try communicating with the adapter
352.                 try
353.                 {
354.                     // If stream is readable
355.                     if (ArduinoStream.CanRead)
356.                     {
357.                         // While stream is readable AND devices are connected
358.                         while (ArduinoStream.CanRead &&
BEATSYNC_GLOBALS.bPaired)
359.                         {
360.                             // While program is passive, publish BPM values
361.                             while (ArduinoStream.CanRead &&
BEATSYNC_GLOBALS.bPaired &&
!BEATSYNC_GLOBALS.bActive)
362.                             {
363.                                 PublishProgress(ArduinoStream.ReadByte());
364.                             }
365.
366.                             // If program became active
367.                             if (ArduinoStream.CanRead &&
BEATSYNC_GLOBALS.bPaired &&
BEATSYNC_GLOBALS.bActive)
368.                             {
369.                                 // Set the first received value as the first ave
rage BPM
370.                                 averageBPM = ArduinoStream.ReadByte();
371.
372.                                 // Run the following command on main UI thread
373.                                 ((Activity)txtBPM.Context).RunOnUiThread(() =>
374.                                 {
375.                                     // Play song with the closest BPM to the ave
rage value
376.                                     Playlist.SetSong(PickSong(averageBPM));
377.                                 });
378.
379.                                 // Nullify summation values
380.                                 updateCounter = summatedBPM = 0;
381.                             }
382.                         }
383.                     }
384.                 }
385.             }
386.         }

```

```

387.
388.                                // While stream is readable AND program is active, keep publishing BPM
389.                                while (ArduinoStream.CanRead &&
390.                                       BEATSYNC_GLOBALS.bPaired &&
391.                                       BEATSYNC_GLOBALS.bActive)
392.                                {
393.                                    PublishProgress(ArduinoStream.ReadByte());
394.                                }
395.                            }
396.                    }
397.            }
398.
399.            // If connection with Arduino was terminated
400.            catch (Java.IO.IOException)
401.            {
402.                return (true);
403.            }
404.
405.            return (false);
406.        }
407.
408.        /// <summary>
409.        /// Updates the BPM label with the data received over bluetooth.
410.        /// If program is active, manages song selection.
411.        /// Labels can be updated here since this procedure is on the UI thread.
412.        /// </summary>
413.        /// <param name="values">New BPM value.</param>
414.        protected override void OnProgressUpdate(params int[] values)
415.        {
416.            // Update BPM label
417.            txtBPM.Text = values[0].ToString();
418.
419.            // Add value to BPM summation
420.            summatedBPM += values[0];
421.
422.            // If enough check have been made AND program is active AND new
            // average differs by at least 10 BPM
423.            // OR song has reached the outro
424.            if ((++updateCounter >= 120 && BEATSYNC_GLOBALS.bActive && System.Math.Abs(averageBPM - (summatedBPM / (float)updateCounter)) > 10) ||
425.                BEATSYNC_GLOBALS.currentSong.CurrentPosition > BEATSYNC_GLOBALS.currentSong.Duration -
426.                (int)(BEATSYNC_GLOBALS.currentSong.Duration / BEATSYNC_CONSTANTS.BEATSYNC_SONGCUE))
427.            {
428.                // If songs are currently not being crossfaded
429.                if (!BEATSYNC_GLOBALS.bCrossfading)
430.                {
431.                    // Update average BPM value
432.                    averageBPM = (int)(summatedBPM / (float)updateCounter);
433.
434.                    // Play song with the closest BPM to current average BPM
435.                    Playlist.SetSong(PickSong(averageBPM));
436.
437.                    // Nullify counter and summation
438.                    updateCounter = summatedBPM = 0;
439.                }

```

```

440.     }
441.
442.     /// <summary>
443.     /// Called on disrupted connection with Arduino adapter.
444.     /// </summary>
445.     /// <param name="bException">If task ended with an exception.</param>
    >
446.     protected override void OnPostExecute(bool bException)
447.     {
448.         // Set program back to passive
449.         BEATSYNC_GLOBALS.bActive = BEATSYNC_GLOBALS.bPaired = false;
450.         SetStatus("Device not paired.", BEATSYNC_CONSTANTS.Colors.Red);
451.
452.         txtBPM.Text = "N/A";
453.         btnMain.SetImageResource(Resource.Drawable.txt_pair);
454.     }
455.
456.     /// <summary>
457.     /// Function is called when device is rotated.
458.     /// </summary>
459.     /// <param name="newConfig">New configuraion.</param>
460.     public override void OnConfigurationChanged(Configuration newConfig)
461.     {
462.         // Execute default configuration commands
463.         base.OnConfigurationChanged(newConfig);
464.
465.         // If device is turned horizontally
466.         if (newConfig.Orientation == Android.Content.Res.Orientation.Landsca
pe)
467.         {
468.             // Hide unneeded views
469.             btnMain.Visibility = ViewStates.Gone;
470.             txtStatus.Visibility = ViewStates.Gone;
471.         }
472.
473.         // If device is turned vertically
474.         else
475.         {
476.             // Show hidden views
477.             btnMain.Visibility = ViewStates.Visible;
478.             txtStatus.Visibility = ViewStates.Visible;
479.         }
480.     }
481.
482.     protected override void OnCreate(Bundle savedInstanceState)
483.     {
484.         // Default creation procedure
485.         base.OnCreate(savedInstanceState);
486.
487.         // Set view to "Main" activity
488.         SetContentView(Resource.Layout.Main);
489.
490.         // If age is stored, load it
491.         if (new Java.IO.File(FilesDir, BEATSYNC_CONSTANTS.BEATSYNC_USERDATA)
.Exists())
492.         {
493.             using (var streamReader = new StreamReader(System.IO.Path.Combin
e(FilesDir.AbsolutePath, BEATSYNC_CONSTANTS.BEATSYNC_USERDATA)))
494.             {
495.                 SetAge((userAge = (char)streamReader.Read()));

```



```

496.         }
497.     }
498.
499.     // Otherwise set to default
500.     else
501.     {
502.         SetAge((userAge = BEATSYNC_CONSTANTS.BEATSYNC_DEFAULTAGE));
503.         using (var streamWriter = new StreamWriter(System.IO.Path.Combine(FilesDir.AbsolutePath, BEATSYNC_CONSTANTS.BEATSYNC_USERDATA)))
504.         {
505.             streamWriter.Write((char)userAge);
506.         }
507.
508.         // Jump to "Settings" screen
509.         StartActivity(typeof(Settings));
510.     }
511.
512.     // If there are analyzed and stored songs, load them
513.     if (new Java.IO.File(FilesDir, BEATSYNC_CONSTANTS.BEATSYNC_USERSONGS).Exists())
514.     {
515.         using (var streamReader = new StreamReader(System.IO.Path.Combine(FilesDir.AbsolutePath, BEATSYNC_CONSTANTS.BEATSYNC_USERSONGS)))
516.         {
517.             while (!streamReader.EndOfStream)
518.             {
519.                 BEATSYNC_GLOBALS.songBPMs.Add(streamReader.ReadLine(), int.Parse(streamReader.ReadLine()));
520.             }
521.         }
522.     }
523.
524.     // If there is a stored playlist, load it
525.     if (new Java.IO.File(FilesDir, BEATSYNC_CONSTANTS.BEATSYNC_USERPLAYLIST).Exists())
526.     {
527.         using (var streamReader = new StreamReader(System.IO.Path.Combine(FilesDir.AbsolutePath, BEATSYNC_CONSTANTS.BEATSYNC_USERPLAYLIST)))
528.         {
529.             while (!streamReader.EndOfStream)
530.             {
531.                 BEATSYNC_GLOBALS.userPlaylist.Add(streamReader.ReadLine());
532.             }
533.         }
534.     }
535.
536.     // Initialize views
537.     btnMain = FindViewById<ImageView>(Resource.Id.btnMain);
538.     txtBPM = FindViewById<TextView>(Resource.Id.BPM);
539.     txtStatus = FindViewById<TextView>(Resource.Id.txtStatus);
540.     btnPlaylist = FindViewById<ImageView>(Resource.Id.imgPlaylist);
541.     txtSong = FindViewById<TextView>(Resource.Id.txtSong);
542.     btnSettings = FindViewById<ImageView>(Resource.Id.imgSettings);
543.
544.     // Set-up adapter and receiver
545.     PhoneAdapter = BluetoothAdapter.DefaultAdapter;
546.     Receiver = new ArduinoBroadcastReceiver();
547.     RegisterReceiver(Receiver, new IntentFilter(BluetoothDevice.ActionFound));
548.

```

```

549.         // Set text colors
550.         FindViewById<TextView>(Resource.Id.txtBPM).SetTextColor(BEATSYNC_CON
    STANTS.Colors.TextGray);
551.         txtSong.SetTextColor(BEATSYNC_CONSTANTS.Colors.TextGray);
552.
553.         // On song title click
554.         txtSong.Click += delegate
555.         {
556.             // If program is passive
557.             if (!BEATSYNC_GLOBALS.bActive)
558.             {
559.                 // If song is being played, pause it
560.                 if (BEATSYNC_GLOBALS.currentSong.IsPlaying)
561.                 {
562.                     BEATSYNC_GLOBALS.currentSong.Pause();
563.                 }
564.
565.                 // Otherwise resume it
566.                 else
567.                 {
568.                     BEATSYNC_GLOBALS.currentSong.Start();
569.                 }
570.             }
571.         };
572.
573.         // On "Playlist" button click
574.         btnPlaylist.Click += delegate
575.         {
576.             // Start "Playlist" activity
577.             StartActivity(typeof(Playlist));
578.         };
579.
580.         // On "Settings" button click
581.         btnSettings.Click += delegate
582.         {
583.             // Start "Settings" activity
584.             StartActivity(typeof(Settings));
585.         };
586.
587.         // If text of the BPM value was changed
588.         txtBPM.AfterTextChanged += delegate
589.         {
590.             // If devices are paired AND new BPM value is above the limit
591.             if (BEATSYNC_GLOBALS.bPaired &&
592.                 int.Parse(txtBPM.Text) > userMaxBPM)
593.             {
594.                 // Set color to red
595.                 txtBPM.SetTextColor(BEATSYNC_CONSTANTS.Colors.Red);
596.
597.                 // If program is active AND music is playing
598.                 if (BEATSYNC_GLOBALS.bActive &&
599.                     BEATSYNC_GLOBALS.currentSong.IsPlaying)
600.                 {
601.                     // Stop the music and notify user of high heart rate
602.                     BEATSYNC_GLOBALS.currentSong.Stop();
603.                     Android.Media.MediaPlayer.Create(this, Android.Net.Uri.P
arase("system/media/audio/ui/LowBattery.ogg")).Start();
604.                 }
605.             }
606.
607.             // If devices are paired AND new BPM value is below the limit

```

```

608.         else if (BEATSYNC_GLOBALS.bPaired &&
609.                 int.Parse(txtBPM.Text) < userMinBPM)
610.         {
611.             // Set color to gray
612.             txtBPM.SetTextColor(BEATSYNC_CONSTANTS.Colors.TextGray);
613.         }
614.
615.         // Otherwise
616.         else
617.         {
618.             // Set color to blue
619.             txtBPM.SetTextColor(BEATSYNC_CONSTANTS.Colors.Blue);
620.
621.             // If program is active AND music is not playing
622.             if (BEATSYNC_GLOBALS.bActive &&
623.                 !BEATSYNC_GLOBALS.currentSong.IsPlaying)
624.             {
625.                 // Resume the song
626.                 BEATSYNC_GLOBALS.currentSong.Start();
627.             }
628.         }
629.     };
630.
631.     // Save default alpha value for the main button
632.     btnMain_DefaultAlpha = btnMain.Alpha;
633.
634.     // On main button click
635.     btnMain.Click += delegate
636.     {
637.         // Lower the alpha value of the view
638.         btnMain.Alpha -= 0.125f;
639.
640.         // If devices are not paired
641.         if (!BEATSYNC_GLOBALS.bPaired)
642.         {
643.             // Disable main button
644.             btnMain.Clickable = false;
645.
646.             // If bluetooth is disabled
647.             if (!PhoneAdapter.IsEnabled)
648.             {
649.                 // Request user to enable bluetooth
650.                 Intent enableBluetooth = new Intent(BluetoothAdapter.Act
ionRequestEnable);
651.                 StartActivityForResult(enableBluetooth, BEATSYNC_CONSTAN
TS.REQUEST_ENABLE_BLUETOOTH);
652.             }
653.
654.             // Otherwise send an already positive request
655.             else
656.             {
657.                 OnActivityResult(BEATSYNC_CONSTANTS.REQUEST_ENABLE_BLUET
OOTH, Result.Ok, new Intent());
658.             }
659.         }
660.
661.         // If devices are paired
662.         else
663.         {
664.             // If program is active
665.             if (BEATSYNC_GLOBALS.bActive)

```

```

666.         {
667.             // Switch program to passive
668.             BEATSYNC_GLOBALS.bActive = false;
669.             btnMain.SetImageResource(Resource.Drawable.icon_play);
670.             SetStatus("Device paired.", BEATSYNC_CONSTANTS.Colors.Te
xtGray);
671.         }
672.
673.         // If program is passive
674.         else
675.         {
676.             // If there are enough songs in the playlist
677.             if (BEATSYNC_GLOBALS.userPlaylist.Count >= BEATSYNC_CONS
TANTS.BEATSYNC_MINPLAYLIST)
678.             {
679.                 // Switch program to active
680.                 BEATSYNC_GLOBALS.bActive = true;
681.                 btnMain.SetImageResource(Resource.Drawable.icon_paus
e);
682.                 SetStatus("Active.", BEATSYNC_CONSTANTS.Colors.TextG
ray);
683.             }
684.
685.             // Otherwise notify user
686.             else
687.             {
688.                 Toast.MakeText(this, "Not enough songs in the Playli
st.", ToastLength.Short).Show();
689.             }
690.         }
691.
692.         // Return to the default alpha value of the main button
693.         btnMain.Alpha = btnMain_DefaultAlpha;
694.     }
695. };
696.
697. // Change status bar color
698. if (Build.VERSION.SdkInt >= BuildVersionCodes.Lollipop)
699. {
700.     Window.ClearFlags(WindowManagerFlags.TranslucentStatus);
701.     Window.AddFlags(WindowManagerFlags.DrawsSystemBarBackgrounds);
702.     Window.SetStatusBarColor(Color.DarkGray);
703. }
704.
705. // Begin analyzing songs in the background
706. new Playlist.SongAnalyzer().ExecuteOnExecutor(AsyncTask.ThreadPoolEx
ecutor, Android.OS.Environment.ExternalStorageDirectory.ToString() + "/Music",
707.     Android.OS.Environment.ExternalStorageDirectory.ToString() + "/Download");
708. }
709.
710. protected override void OnDestroy()
711. {
712.     // Execute default "OnDestroy" commands
713.     base.OnDestroy();
714.
715.     // Unregister the arduino receiver
716.     UnregisterReceiver(Receiver);
717. }
718. }
719. }

```

קוד הפרוייקט – BeatSync – Main.xml

```
1. <?xml version="1.0" encoding="utf-8"?>
2. <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
3.     android:orientation="vertical"
4.     android:layout_width="match_parent"
5.     android:layout_height="match_parent"
6.     android:background="@drawable/background"
7.     android:minWidth="25px"
8.     android:minHeight="25px">
9.     <RelativeLayout
10.         android:layout_width="match_parent"
11.         android:layout_height="wrap_content"
12.         android:layout_marginLeft="10dp"
13.         android:layout_marginRight="10dp"
14.         android:layout_marginTop="10dp">
15.         <ImageView
16.             android:background="@drawable/button"
17.             android:src="@drawable/icon_note"
18.             android:layout_width="50dp"
19.             android:layout_height="50dp"
20.             android:alpha="0.875"
21.             android:id="@+id/imgPlaylist" />
22.         <ImageView
23.             android:background="@drawable/button"
24.             android:src="@drawable/icon_settings"
25.             android:layout_width="50dp"
26.             android:layout_height="50dp"
27.             android:layout_alignParentRight="true"
28.             android:alpha="0.875"
29.             android:id="@+id/imgSettings" />
30.     </RelativeLayout>
31.     <TextView
32.         android:text="N/A"
33.         android:layout_width="match_parent"
34.         android:layout_height="wrap_content"
35.         android:layout_marginTop="-25dp"
36.         android:gravity="center"
37.         android:textColor="#3498db"
38.         android:id="@+id/BPM"
39.         android:textSize="100sp" />
40.     <TextView
41.         android:text="BPM"
42.         android:layout_width="match_parent"
43.         android:layout_height="wrap_content"
44.         android:id="@+id/txtBPM"
45.         android:gravity="center"
46.         android:layout_marginBottom="10dp"
47.         android:textSize="25sp" />
48.     <TextView
49.         android:text="▶ "
50.         android:layout_width="match_parent"
51.         android:layout_height="wrap_content"
52.         android:id="@+id/txtSong"
53.         android:textSize="17.5sp"
54.         android:gravity="center"
55.         android:layout_margin="10dp" />
56.     <RelativeLayout
57.         android:layout_width="match_parent"
58.         android:layout_height="wrap_content"
```

```

59.         android:layout_marginLeft="10dp"
60.         android:layout_marginRight="10dp"
61.         android:layout_marginTop="10dp"
62.         android:layout_marginBottom="30dp">
63.         <ImageView
64.             android:background="@drawable/button"
65.             android:src="@drawable/txt_pair"
66.             android:layout_width="235dp"
67.             android:layout_height="235dp"
68.             android:layout_marginBottom="-20dp"
69.             android:id="@+id/btnMain"
70.             android:alpha="0.875"
71.             android:layout_centerHorizontal="true"
72.             android:layout_above="@+id/txtStatus" />
73.         <TextView
74.             android:text="Device not paired."
75.             android:layout_width="match_parent"
76.             android:layout_height="wrap_content"
77.             android:textColor="#800000"
78.             android:id="@+id/txtStatus"
79.             android:layout_alignParentBottom="true"
80.             android:gravity="center"
81.             android:textSize="25sp" />
82.     </RelativeLayout>
83. </LinearLayout>

```

קוד הפרוייקט – BeatSync – AndroidManifest.xml

```

1. <?xml version="1.0" encoding="utf-8"?>
2. <manifest xmlns:android="http://schemas.android.com/apk/res/android" package="BeatSync.
   BeatSync" android:versionCode="1" android:versionName="1.0" android:installLocation="au
   to">
3.     <uses-sdk android:minSdkVersion="21" />
4.     <uses-permission android:name="android.permission.BLUETOOTH_ADMIN" />
5.     <uses-permission android:name="android.permission.BLUETOOTH" />
6.     <uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE" />
7.     <application android:allowBackup="true" android:label="@string/app_name" android:th
   eme="@android:style/Theme.DeviceDefault.Light.NoActionBar" android:icon="@drawable/icon
   _app"></application>
8. </manifest>

```

Playlist.cs – BeatSync – קוד הפרוייקט

```
1. using System.IO;
2. using Android.App;
3. using Android.OS;
4. using Android.Graphics;
5. using Android.Runtime;
6. using Android.Views;
7. using Android.Widget;
8.
9. namespace BeatSync
10. {
11.     [Activity(Label = "Playlist")]
12.
13.     public class Playlist : Activity
14.     {
15.         // Current activity views
16.         private static ListView    songList;
17.         private static ProgressBar bpmProgressBar;
18.         private static ImageView   btnBack;
19.
20.         /// <summary>
21.         /// Tries to save the playlist in its current state to disk.
22.         /// </summary>
23.         /// <returns>True if saved successfully, false otherwise.</returns>
24.         public static bool SavePlaylist()
25.         {
26.             // Try storing playlist on disk
27.             try
28.             {
29.                 using (var streamWriter = new StreamWriter(System.IO.Path.Combine(Applic
30.                 {
31.                     foreach (string songPath in BEATSYNC_GLOBALS.userPlaylist)
32.                     {
33.                         streamWriter.WriteLine(songPath);
34.                     }
35.                 }
36.
37.                 return (true);
38.             }
39.
40.             // If file is unavailable for writing, don't save
41.             catch (System.IO.IOException)
42.             {
43.                 return (false);
44.             }
45.         }
46.
47.         /// <summary>
48.         /// Receives a path to the next song and crossfades the current song with the n
49.         /// </summary>
50.         protected class Crossfade : AsyncTask<string, bool, bool>
51.         {
52.             protected override void OnPreExecute()
53.             {
54.                 // Set crossfading boolean
55.                 BEATSYNC_GLOBALS.bCrossfading = true;
```

```

56.         }
57.
58.         protected override bool RunInBackground(params string[] @params)
59.         {
60.             // Create a next song player and a crossfade timer
61.             Android.Media.MediaPlayer nextSong = Android.Media.MediaPlayer.Create(A
        ndroid.App.Application.Context,
62.                                     A
        ndroid.Net.Uri.Parse(@params[0]));
63.             System.Diagnostics.Stopwatch faderTimer = new System.Diagnostics.Stopwa
        tch();
64.
65.             // Set up the next song
66.             nextSong.SeekTo((int)(nextSong.Duration / BEATSYNC_CONSTANTS.BEATSYNC_S
        ONGCUE));
67.             nextSong.SetVolume(0.0f, 0.0f);
68.
69.             // Begin playback and crossfade
70.             faderTimer.Start();
71.             nextSong.Start();
72.
73.             // Crossfade until time limit has passed
74.             while (faderTimer.ElapsedMilliseconds < 5000)
75.             {
76.                 float currentVolume = faderTimer.ElapsedMilliseconds / 5000f;
77.                 nextSong.SetVolume(currentVolume, currentVolume);
78.                 BEATSYNC_GLOBALS.currentSong.SetVolume(1f - currentVolume, 1f -
        currentVolume);
79.             }
80.
81.             // Stop the previous song
82.             BEATSYNC_GLOBALS.currentSong.Stop();
83.
84.             // Next song is now the currently playing song
85.             BEATSYNC_GLOBALS.currentSong = nextSong;
86.
87.             return (true);
88.         }
89.
90.         protected override void OnPostExecute(bool result)
91.         {
92.             // Set crossfading boolean
93.             BEATSYNC_GLOBALS.bCrossfading = false;
94.         }
95.     }
96.
97.     /// <summary>
98.     /// Crossfades to the new song and updates the song label.
99.     /// </summary>
100.    /// <param name="path">Path to a new song.</param>
101.    public static void SetSong(string path)
102.    {
103.        new Crossfade().Execute(path);
104.        BEATSYNC_GLOBALS.currentlyPlaying = path;
105.        MainActivity.SetSongLabel(BEATSYNC_GLOBALS.songTitles[path]);
106.    }
107.
108.    /// <summary>
109.    /// Background task that analyzes every .WAV file in the given paths.
110.    /// Task calculates the BPM of every "wave" file not previously analyzed
    .

```



```

111.         /// Task then saves the calculation to song dictionary.
112.         /// </summary>
113.         public class SongAnalyzer : AsyncTask<string, bool, bool>
114.         {
115.             /// <summary>
116.             /// Receives the paths to look for .WAV files in.
117.             /// Analyzes each "wave" file and saves to list of songs.
118.             /// </summary>
119.             /// <param name="params">Paths to .WAV files.</param>
120.             /// <returns>Returns true.</returns>
121.             protected override bool RunInBackground(params string[] @params)
122.             {
123.                 // Iterate over each given directory
124.                 for (int nIndex = 0;
125.                     nIndex < @params.Length;
126.                     nIndex++)
127.                 {
128.                     // For each object in the directory
129.                     foreach (Java.IO.File File in new Java.IO.File(@params[nIndex]).ListFiles())
130.                     {
131.                         // If object is a .WAV file AND its duration will suffice for BPM calculation
132.                         if (File.IsFile &&
133.                             WAV.IsWAV(File.Path) &&
134.                             Android.Media.MediaPlayer.Create(Application.Context, Android.Net.Uri.Parse(File.Path)).Duration >= WAV.BPM_CALCULATION_DURATION_MINIMUM * 1000)
135.                         {
136.                             // If BPM for this song was NOT previously calculated
137.                             if (!BEATSYNC_GLOBALS.songBPMs.ContainsKey(File.Path))
138.                             {
139.                                 // Load current song to the codec
140.                                 WAV waveFile = new WAV(File.Path);
141.
142.                                 // Calculate BPM and add to song dictionary
143.                                 BEATSYNC_GLOBALS.songBPMs.Add(waveFile.Path, waveFile.BPM);
144.                                 BEATSYNC_GLOBALS.userPlaylist.Add(waveFile.Path);
145.
146.                                 // Write calculations to disk
147.                                 using (var songWriter = new StreamWriter(System.IO.Path.Combine(Application.Context.FilesDir.AbsolutePath, BEATSYNC_CONSTANTS.BEATSYNC_USERSONGS), true))
148.                                 {
149.                                     songWriter.WriteLine(waveFile.Path);
150.                                     songWriter.WriteLine(waveFile.BPM.ToString());
151.                                 }
152.
153.                                 // Add new song to playlist
154.                                 using (var playlistWriter = new StreamWriter(System.IO.Path.Combine(Application.Context.FilesDir.AbsolutePath, BEATSYNC_CONSTANTS.BEATSYNC_USERPLAYLIST), true))
155.                                 {
156.                                     playlistWriter.WriteLine(waveFile.Path);
157.                                 }
158.                             }

```

```

159.
160.                // Set title for the current song
161.                BEATSYNC_GLOBALS.songTitles.Add(File.Path, File.Name
.Substring(0, File.Name.Length -
4) + " [" + BEATSYNC_GLOBALS.songBPMs[File.Path] + " BPM]");
162.
163.                // Add current song to list of available songs
164.                PublishProgress(true);
165.                System.Threading.Thread.Sleep(50);
166.            }
167.        }
168.    }
169.
170.    return (true);
171.}
172.
173.    /// <summary>
174.    /// Receives a single ArrayAdapter with currently available songs.
175.    /// Sets the received adapter as the main song list adapter.
176.    /// </summary>
177.    /// <param name="values">Adapter with available songs.</param>
178.    protected override void OnProgressUpdate(params bool[] values)
179.    {
180.        // Refresh list of songs on new song addition
181.        RefreshSongList();
182.    }
183.
184.    protected override void OnPostExecute(bool result)
185.    {
186.        // If progress bar is initialized, hide it
187.        if (bpmProgressBar != null)
188.        {
189.            bpmProgressBar.Visibility = ViewStates.Gone;
190.        }
191.
192.        // Save user playlist
193.        SavePlaylist();
194.        BEATSYNC_GLOBALS.bSongsAnalyzed = true;
195.    }
196.}
197.
198.    /// <summary>
199.    /// Refreshes the list of analyzed and stored songs.
200.    /// </summary>
201.    private static void RefreshSongList()
202.    {
203.        // If stored list of calculated songs exists AND song list view is i
nitialized
204.        if (new Java.IO.File(Application.Context.FilesDir, BEATSYNC_CONSTANT
S.BEATSYNC_USERSONGS).Exists() &&
205.            songList != null)
206.        {
207.            // Create a new list and add the song paths to the list
208.            JavaList<string> analyzedSongs = new JavaList<string>();
209.            using (var songReader = new StreamReader(System.IO.Path.Combine(
Application.Context.FilesDir.AbsolutePath, BEATSYNC_CONSTANTS.BEATSYNC_USERSONGS), true
))
210.            {
211.                // Read until the end of file
212.                while (!songReader.EndOfStream)
213.                {

```

```

214.                // Save current song path
215.                string currentPath = songReader.ReadLine();
216.
217.                // If path is located in the dictionary of song titles,
add to analyzed songs
218.                if (BEATSYNC_GLOBALS.songTitles.ContainsKey(currentPath)
)
219.                {
220.                    analyzedSongs.Add(BEATSYNC_GLOBALS.songTitles[curren
tPath]);
221.                }
222.
223.                // Skip the BPM value for this song
224.                songReader.ReadLine();
225.            }
226.        }
227.
228.        // Set-
up custom adapter with calculated songs to song listview
229.        songList.Adapter = new ArrayAdapter(songList.Context,
Android.Resource.Layout.Simp
230.        leListItemMultipleChoice,
analyzedSongs);
231.
232.
233.        // Check song if it's in the playlist
234.        for (int nSongIndex = 0;
235.            nSongIndex < songList.Count;
236.            nSongIndex++)
237.        {
238.            songList.SetItemChecked(nSongIndex, BEATSYNC_GLOBALS.userPla
ylist.Contains(MainActivity.GetKeyByValue(BEATSYNC_GLOBALS.songTitles, songList.GetItem
AtPosition(nSongIndex).ToString())));
239.        }
240.    }
241.}
242.
243.    protected override void OnCreate(Bundle savedInstanceState)
244.    {
245.        // Default creation procedure
246.        base.OnCreate(savedInstanceState);
247.
248.        // Set view to "Playlist" activity
249.        SetContentView(Resource.Layout.Playlist);
250.
251.        // Initialize views
252.        bpmProgressBar = FindViewById<ProgressBar>(Resource.Id.pbProgressBa
r);
253.        btnBack = FindViewById<ImageView>(Resource.Id.imgBack);
254.
255.        // Set-up the song list
256.        songList = FindViewById<ListView>(Resource.Id.lvSongs);
257.        songList.ChoiceMode = ChoiceMode.Multiple;
258.
259.        // Change the color of the title text
260.        FindViewById<TextView>(Resource.Id.txtPlaylist).SetTextColor(BEATSYN
C_CONSTANTS.Colors.TextGray);
261.
262.        // Finish activity on "BACK" button click
263.        btnBack.Click += delegate
264.        {
265.            btnBack.Alpha = 0.55f;

```

```

266.             Finish();
267.         };
268.
269.         // Add or remove a song from the playlist on song click
270.         songList.ItemClick += (sender, e) =>
271.         {
272.             // If item is now checked, add song to playlist
273.             if (songList.IsItemChecked(e.Position))
274.             {
275.                 BEATSYNC_GLOBALS.userPlaylist.Add(MainActivity.GetKeyByValue
276. (BEATSYNC_GLOBALS.songTitles, songList.GetItemAtPosition(e.Position).ToString()));
277.             }
278.             // If item is now unchecked
279.             else
280.             {
281.                 // If playlist has enough songs to go on, remove song from p
282.                 if (BEATSYNC_GLOBALS.userPlaylist.Count > BEATSYNC_CONSTANTS
283. .BEATSYNC_MINPLAYLIST)
284.                 {
285.                     BEATSYNC_GLOBALS.userPlaylist.Remove(MainActivity.GetKey
286. ByValue(BEATSYNC_GLOBALS.songTitles, songList.GetItemAtPosition(e.Position).ToString())
287. );
288.                 }
289.                 // Otherwise check the song back
290.                 else
291.                 {
292.                     songList.SetItemChecked(e.Position, true);
293.                     Toast.MakeText(this, "At least " + BEATSYNC_CONSTANTS.BE
294. ATSYNC_MINPLAYLIST + " songs are required.", ToastLength.Short).Show();
295.                 }
296.             }
297.         };
298.
299.         // Preview a long-clicked song
300.         songList.ItemLongClick += (sender, e) =>
301.         {
302.             // If program is passive
303.             if (!BEATSYNC_GLOBALS.bActive)
304.             {
305.                 // Preview chosen song
306.                 SetSong(MainActivity.GetKeyByValue(BEATSYNC_GLOBALS.songTitl
307. es, songList.GetItemAtPosition(e.Position).ToString()));
308.             }
309.         };
310.
311.         // If songs were successfully analyzed, refresh the song list
312.         if (BEATSYNC_GLOBALS.bSongsAnalyzed == true)
313.         {
314.             RefreshSongList();
315.             bpmProgressBar.Visibility = ViewStates.Gone;
316.         }
317.
318.         // Change status bar color
319.         if (Build.VERSION.SdkInt >= BuildVersionCodes.Lollipop)
320.         {

```

```

320.             Window.ClearFlags(WindowManagerFlags.TranslucentStatus);
321.             Window.AddFlags(WindowManagerFlags.DrawsSystemBarBackgrounds);
322.             Window.SetStatusBarColor(Color.DimGray);
323.         }
324.
325.         // If program is passive
326.         if (!BEATSYNC_GLOBALS.bActive)
327.         {
328.             // Notify user of preview feature
329.             Toast.MakeText(this, "Hold to preview.", ToastLength.Long).Show(
330.         );
331.         }
332.     }
333. }

```

Playlist.xml – BeatSync – קוד הפרוייקט

```
1. <?xml version="1.0" encoding="utf-8"?>
2. <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
3.     android:orientation="vertical"
4.     android:layout_width="match_parent"
5.     android:layout_height="match_parent"
6.     android:minWidth="25px"
7.     android:minHeight="25px">
8.     <RelativeLayout
9.         android:layout_width="match_parent"
10.        android:layout_height="35dp"
11.        android:layout_marginTop="10dp"
12.        android:layout_marginBottom="10dp"
13.        android:layout_marginLeft="10dp">
14.        <ImageView
15.            android:src="@drawable/icon_back"
16.            android:layout_height="match_parent"
17.            android:layout_width="30dp"
18.            android:id="@+id/imgBack" />
19.        <TextView
20.            android:text="Playlist"
21.            android:layout_height="match_parent"
22.            android:layout_width="wrap_content"
23.            android:layout_centerHorizontal="true"
24.            android:textSize="25sp"
25.            android:id="@+id/txtPlaylist" />
26.    </RelativeLayout>
27.    <ListView
28.        android:layout_width="match_parent"
29.        android:layout_height="wrap_content"
30.        android:id="@+id/lvSongs" />
31.    <ProgressBar
32.        android:layout_width="match_parent"
33.        android:layout_height="wrap_content"
34.        android:layout_marginTop="10dp"
35.        android:indeterminate="true"
36.        android:id="@+id/pbProgressBar" />
37. </LinearLayout>
```

Settings.cs – BeatSync – קוד הפרוייקט

```
1. using System.IO;
2. using Android.App;
3. using Android.OS;
4. using Android.Views;
5. using Android.Widget;
6. using Android.Graphics;
7.
8. namespace BeatSync
9. {
10.     [Activity(Label = "Settings")]
11.     public class Settings : Activity
12.     {
13.         // Current activity views
14.         private static NumberPicker agePicker;
15.         private static TextView btnClearCache;
16.         private static TextView btnDone;
17.         private static TextView txtRange;
18.
19.         /// <summary>
20.         /// Updates user age with the current number picker value.
21.         /// Updates the safe BPM range based on user's age.
22.         /// </summary>
23.         private static void UpdateAge()
24.         {
25.             int[] safeRange = MainActivity.SetAge(agePicker.Value);
26.             txtRange.Text = "Safe BPM range: " + safeRange[0] + " - " + safeRange[1];
27.         }
28.
29.         protected override void OnCreate(Bundle savedInstanceState)
30.         {
31.             // Default creation procedure
32.             base.OnCreate(savedInstanceState);
33.
34.             // Set view to "Settings" activity
35.             SetContentView(Resource.Layout.Settings);
36.
37.             // Initialize range textview
38.             txtRange = FindViewById<TextView>(Resource.Id.txtRange);
39.
40.             // Set-up the age number picker
41.             agePicker = FindViewById<NumberPicker>(Resource.Id.npAgePicker);
42.             agePicker.MinValue = BEATSYNC_CONSTANTS.BEATSYNC_MINIMUMAGE;
43.             agePicker.MaxValue = BEATSYNC_CONSTANTS.BEATSYNC_MAXIMUMAGE;
44.             agePicker.WrapSelectorWheel = false;
45.             agePicker.DescendantFocusability = DescendantFocusability.BlockDescendants;
46.
47.             // If user age is stored, load it
48.             if (new Java.IO.File(Application.Context.FilesDir, BEATSYNC_CONSTANTS.BEATS
49. YNC_USERDATA).Exists())
50.             {
51.                 using (var streamReader = new StreamReader(System.IO.Path.Combine(Files
52. Dir.AbsolutePath, BEATSYNC_CONSTANTS.BEATSYNC_USERDATA)))
53.                 {
54.                     agePicker.Value = (char)streamReader.Read();
55.                 }
56.             }
57.         }
58.     }
59. }
```

```

56.         // Otherwise set to default
57.     else
58.     {
59.         agePicker.Value = BEATSYNC_CONSTANTS.BEATSYNC_DEFAULTAGE;
60.     }
61.
62.     // Set-up the "CLEAR LOCAL CACHE" button
63.     btnClearCache = FindViewById<TextView>(Resource.Id.btnClearCache);
64.     btnClearCache.SetBackgroundColor(Color.WhiteSmoke);
65.
66.     // Set-up the "DONE" button
67.     btnDone = FindViewById<TextView>(Resource.Id.btnDone);
68.     btnDone.SetTextColor(Color.White);
69.     btnDone.SetBackgroundColor(BEATSYNC_CONSTANTS.Colors.Blue);
70.     btnDone.Alpha = 0.85f;
71.
72.     // Finish activity on "DONE" button click
73.     btnDone.Click += delegate
74.     {
75.         btnDone.Alpha = 0.75f;
76.         Finish();
77.     };
78.
79.     // Clear cache and restart the app
80.     btnClearCache.LongClick += delegate
81.     {
82.         // Delete age
83.         if (new Java.IO.File(FileDir, BEATSYNC_CONSTANTS.BEATSYNC_USERDATA).Ex
ists())
84.         {
85.             new Java.IO.File(FileDir, BEATSYNC_CONSTANTS.BEATSYNC_USERDATA).De
lete();
86.         }
87.
88.         // Delete calculated songs list
89.         if (new Java.IO.File(FileDir, BEATSYNC_CONSTANTS.BEATSYNC_USERSONGS).E
xists())
90.         {
91.             new Java.IO.File(FileDir, BEATSYNC_CONSTANTS.BEATSYNC_USERSONGS).D
elete();
92.         }
93.
94.         // Delete playlist info
95.         if (new Java.IO.File(FileDir, BEATSYNC_CONSTANTS.BEATSYNC_USERPLAYLIST
).Exists())
96.         {
97.             new Java.IO.File(FileDir, BEATSYNC_CONSTANTS.BEATSYNC_USERPLAYLIST
).Delete();
98.         }
99.
100.        // Restart the application
101.        Process.KillProcess(Process.MyPid());
102.    };
103.
104.    // Update user age with the new number picker value
105.    agePicker.ValueChanged += delegate
106.    {
107.        // Store age on disk
108.        using (var streamWriter = new StreamWriter(System.IO.Path.Combine
e(FileDir.AbsolutePath, BEATSYNC_CONSTANTS.BEATSYNC_USERDATA)))
109.        {

```



```

110.                streamWriter.Write((char)agePicker.Value);
111.            }
112.
113.            // Update user age
114.            UpdateAge();
115.        };
116.
117.        // Update current user age
118.        UpdateAge();
119.
120.        // Change status bar color
121.        if (Build.VERSION.SdkInt >= BuildVersionCodes.Lollipop)
122.        {
123.            Window.ClearFlags(WindowManagerFlags.TranslucentStatus);
124.            Window.AddFlags(WindowManagerFlags.DrawsSystemBarBackgrounds);
125.            Window.SetStatusBarColor(BEATSYNC_CONSTANTS.Colors.Blue);
126.        }
127.    }
128. }
129.

```

Settings.xml – BeatSync – קוד הפרוייקט

```
1. <?xml version="1.0" encoding="utf-8"?>
2. <LinearLayout xmlns:p1="http://schemas.android.com/apk/res/android"
3.     p1:orientation="vertical"
4.     p1:minWidth="25px"
5.     p1:minHeight="25px"
6.     p1:layout_width="match_parent"
7.     p1:layout_height="match_parent">
8.     <TextView
9.         p1:text="Pick your age"
10.        p1:layout_width="match_parent"
11.        p1:layout_height="wrap_content"
12.        p1:gravity="center"
13.        p1:layout_marginLeft="10dp"
14.        p1:layout_marginRight="10dp"
15.        p1:layout_marginTop="25dp"
16.        p1:textSize="22sp" />
17.     <NumberPicker
18.         p1:layout_width="match_parent"
19.         p1:layout_height="wrap_content"
20.         p1:id="@+id/npAgePicker"
21.         p1:minWidth="25px"
22.         p1:minHeight="25px" />
23.     <RelativeLayout
24.         p1:minWidth="25px"
25.         p1:minHeight="25px"
26.         p1:layout_width="match_parent"
27.         p1:layout_height="wrap_content">
28.         <TextView
29.             p1:text="Safe BPM range: "
30.             p1:layout_width="match_parent"
31.             p1:layout_height="wrap_content"
32.             p1:layout_margin="15dp"
33.             p1:gravity="center"
34.             p1:layout_centerHorizontal="true"
35.             p1:layout_alignParentTop="true"
36.             p1:textSize="18sp"
37.             p1:id="@+id/txtRange" />
38.         <TextView
39.             p1:text="CLEAR LOCAL CACHE"
40.             p1:layout_width="match_parent"
41.             p1:layout_height="wrap_content"
42.             p1:minHeight="50dp"
43.             p1:gravity="center"
44.             p1:layout_centerHorizontal="true"
45.             p1:layout_above="@+id/btnDone"
46.             p1:textSize="30sp"
47.             p1:id="@+id/btnClearCache" />
48.         <TextView
49.             p1:text="DONE"
50.             p1:layout_width="match_parent"
51.             p1:layout_height="wrap_content"
52.             p1:minHeight="50dp"
53.             p1:gravity="center"
54.             p1:layout_centerHorizontal="true"
55.             p1:layout_alignParentBottom="true"
56.             p1:textSize="30sp"
57.             p1:id="@+id/btnDone" />
58.     </RelativeLayout> </LinearLayout>
```

BEATSYNC_VARIABLES.cs – BeatSync – קוד הפרוייקט

```

1. using System.Collections.Generic;
2. using Android.Runtime;
3. using Android.Graphics;
4. using Android.Media;
5. using Java.Util;
6.
7. namespace BeatSync
8. {
9.     /// <summary>
10.    /// Application globals
11.    /// </summary>
12.    class BEATSYNC_GLOBALS
13.    {
14.        /// Music related variables
15.        public static Dictionary<string, int> songBPMs = new Dictionary<string, int>();
16.        public static Dictionary<string, string> songTitles = new Dictionary<string, string>();
17.        public static JavaList<string> userPlaylist = new JavaList<string>();
18.        public static MediaPlayer currentSong = new MediaPlayer();
19.        public static string currentlyPlaying = null;
20.        public static bool bCrossfading = false;
21.        public static bool bSongsAnalyzed = false;
22.
23.        /// Application status variables
24.        public static bool bPaired = false;
25.        public static bool bActive = false;
26.    }
27.
28.    /// <summary>
29.    /// Application constants.
30.    /// </summary>
31.    class BEATSYNC_CONSTANTS
32.    {
33.        /// <summary>
34.        /// Colors designed for BeatSync application.
35.        /// </summary>
36.        public class Colors
37.        {
38.            public static Color Gray { get; } = Color.ParseColor("#7f8c8d");
39.            public static Color Red { get; } = Color.ParseColor("#800000");
40.            public static Color Blue { get; } = Color.ParseColor("#3498db");
41.            public static Color TextGray { get; } = Color.DimGray;
42.        }
43.
44.        /// Application constants
45.        public static readonly string BEATSYNC_APPLICATION_NAME = "BeatSync";
46.        public static readonly int BEATSYNC_NOTIFICATION_ID = 7;
47.        public static readonly int BEATSYNC_DEFAULTAGE = 20;
48.        public static readonly int BEATSYNC_MINIMUMAGE = 16;
49.        public static readonly int BEATSYNC_MAXIMUMAGE = 79;
50.        public static readonly int BEATSYNC_MAXBPM = 220;
51.        public static readonly int BEATSYNC_MINPLAYLIST = 2;
52.        public static readonly float BEATSYNC_SONGCUE = 5.5f;
53.        public static readonly string BEATSYNC_USERDATA = "userdata.dat";

```

```

54.         public static readonly string BEATSYNC_USERSONGS = "usersongs.dat";
55.         public static readonly string BEATSYNC_USERPLAYLIST = "userplaylist.dat";
56.         public static readonly int REQUEST_ENABLE_BLUETOOTH = 1;
57.         public static readonly string ARDUINO_ADDRESS = "98:D3:32:31:17:C";
58.         public static readonly string ARDUINO_PAIRING_PIN = "1234";
59.         public static readonly UUID ARDUINO_UUID = UUID.FromString("00001101-0000-1000-8000-00805f9b34fb");
60.     }
61. }

```

קוד הפרוייקט – WAVE Codec

```
1. using System;
2. using System.Text;
3. using System.IO;
4. using System.Linq;
5. using System.Collections.Generic;
6. using System.Numerics;
7.
8. namespace BeatSync
9. {
10.     /// <summary>
11.     /// Resources used by the codec to encode/decode bitstreams.
12.     /// Resources include various constants and helping functions.
13.     /// </summary>
14.     class Resources
15.     {
16.         public class FourierTransform
17.         {
18.             /// <summary>
19.             /// Computes a Discrete Fourier Transform over given signal samples.
20.             /// Algorithm solves DFT formula over each sample:
21.             ///
22.             ///  $F(k) = 1 / \sqrt{N} * \sum T(n) * e^{(-2 * \pi * n * k / N)}$ 
23.             ///
24.             /// Where:
25.             /// F(k) - frequency bin of a sample;
26.             /// T(n) - current sample;
27.             /// n - sample index;
28.             /// k - frequency index;
29.             /// N - amount of samples in a given signal.
30.             ///
31.             /// Complexity:  $(n^2)$ 
32.             ///
33.             /// </summary>
34.             /// <param name="Signal">Array of amplitude samples of a signal.</param>
35.             /// <returns>Array of frequency bins of a signal.</returns>
36.             public static Complex[] DFT(Complex[] Signal)
37.             {
38.                 // Define formula variables
39.                 int N = Signal.Length, n, k;
40.                 Complex[] Spectrum = new Complex[N];
41.
42.                 // Iterate over signal samples
43.                 for (n = 0;
44.                     n < N;
45.                     n++)
46.                 {
47.                     // Define frequency sigma variable
48.                     Complex cpxFrequencySigma = new Complex(0, 0);
49.
50.                     // Calculate frequency sigma over each sample, per sample
51.                     for (k = 0;
52.                         k < N;
53.                         k++)
54.                     {
55.                         cpxFrequencySigma += Signal[k] * Complex.Exp(new Complex(0, -
56.                             2 * Math.PI * n * k / N));
57.                     }
58.                 }
59.             }
60.         }
61.     }
62. }
```

```

58.
59.         // Store calculated result
60.         Spectrum[n] = (float)1 / Math.Sqrt(N) * cpxFrequencySigma;
61.     }
62.
63.     // Return the spectrum
64.     return (Spectrum);
65. }
66.
67.     /// <summary>
68.     /// More efficient implementation of DFT.
69.     /// THIS CODE IS AN OPEN-SOURCE IMPLEMENTATION OF FFT AND WAS
70.     /// NOT WRITTEN FOR BEATSYNC.
71.     /// </summary>
72.     /// <param name="dir">FFT direction. 1 for FFT, -
1 for Inverse FFT.</param>
73.     /// <param name="m">Power of 2 that is the sample amount.</param>
74.     /// <param name="x">Real part of the samples.</param>
75.     /// <param name="y">Imaginary part of the samples.</param>
76.     public static void FFT(short dir, int m, double[] x, double[] y)
77.     {
78.         int n, i, i1, j, k, i2, l, l1, l2;
79.         double c1, c2, tx, ty, t1, t2, u1, u2, z;
80.
81.         // Calculate the number of points
82.
83.         n = 1;
84.
85.         for (i = 0; i < m; i++)
86.             n *= 2;
87.
88.         // Do the bit reversal
89.
90.         i2 = n >> 1;
91.         j = 0;
92.         for (i = 0; i < n - 1; i++)
93.         {
94.             if (i < j)
95.             {
96.                 tx = x[i];
97.                 ty = y[i];
98.                 x[i] = x[j];
99.                 y[i] = y[j];
100.                 x[j] = tx;
101.                 y[j] = ty;
102.             }
103.             k = i2;
104.
105.             while (k <= j)
106.             {
107.                 j -= k;
108.                 k >>= 1;
109.             }
110.
111.             j += k;
112.         }
113.
114.         // Compute the FFT
115.
116.         c1 = -1.0;
117.         c2 = 0.0;

```

```

118.         l2 = 1;
119.
120.         for (l = 0; l < m; l++)
121.         {
122.             l1 = l2;
123.             l2 <= 1;
124.             u1 = 1.0;
125.             u2 = 0.0;
126.
127.             for (j = 0; j < l1; j++)
128.             {
129.                 for (i = j; i < n; i += l2)
130.                 {
131.                     i1 = i + l1;
132.                     t1 = u1 * x[i1] - u2 * y[i1];
133.                     t2 = u1 * y[i1] + u2 * x[i1];
134.                     x[i1] = x[i] - t1;
135.                     y[i1] = y[i] - t2;
136.                     x[i] += t1;
137.                     y[i] += t2;
138.                 }
139.
140.                 z = u1 * c1 - u2 * c2;
141.                 u2 = u1 * c2 + u2 * c1;
142.                 u1 = z;
143.             }
144.
145.             c2 = Math.Sqrt((1.0 - c1) / 2.0);
146.
147.             if (dir == 1)
148.                 c2 = -c2;
149.
150.             c1 = Math.Sqrt((1.0 + c1) / 2.0);
151.         }
152.
153.         // Scaling for forward transform
154.
155.         if (dir == 1)
156.         {
157.             for (i = 0; i < n; i++)
158.             {
159.                 x[i] /= n;
160.                 y[i] /= n;
161.             }
162.         }
163.     }
164.
165.     /// <summary>
166.     /// Computes an Inverse Discrete Fourier Transform over given signal
    samples.
167.     /// Algorithm solves Inverse DFT formula over each sample:
168.     ///
169.     ///  $T(n) = 1 / \text{Sqrt}(N) * \sum F(k) * e^{(2 * \text{PI} * n * k / N)}$ 
170.     ///
171.     /// Where:
172.     /// F(k) - frequency bin of a sample;
173.     /// T(n) - current sample;
174.     /// n - sample index;
175.     /// k - frequency index;
176.     /// N - amount of samples in a given signal.
177.     ///

```

```

178.          /// Complexity: (n^2)
179.          ///
180.          /// </summary>
181.          /// <param name="Spectrum">Array of frequency bins of a signal.</par
am>
182.          /// <returns>Array of amplitude samples of a signal.</returns>
183.          public static Complex[] InverseDFT(Complex[] Spectrum)
184.          {
185.              // Define formula variables
186.              int N = Spectrum.Length, n, k;
187.              Complex[] Signal = new Complex[N];
188.
189.              // Iterate over frequency bins
190.              for (n = 0;
191.                  n < N;
192.                  n++)
193.              {
194.                  // Define sample sigma variable
195.                  Complex cpxSample = new Complex(0, 0);
196.
197.                  // Calculate sigma
198.                  for (k = 0;
199.                      k < N;
200.                      k++)
201.                  {
202.                      cpxSample += Spectrum[k] * Complex.Exp(new Complex(0, 2
* Math.PI * n * k / N));
203.                  }
204.
205.                  // Store calculated result
206.                  Signal[n] = (float)1 / Math.Sqrt(N) * cpxSample;
207.              }
208.
209.              // Return the signal
210.              return (Signal);
211.          }
212.
213.      }
214.
215.      // Bit-depths supported by this codec
216.      public static readonly int[] ALLOWED_BITDEPTHS =
217.      {
218.          sizeof(short) * 8,
219.          sizeof(float) * 8,
220.      };
221.
222.      // Max amplitude for each supported bit-depth
223.      public static readonly Dictionary<Type, int> MAX_AMPLITUDE = new Dictio
nary<Type, int>
224.      {
225.          { typeof(short[]), short.MaxValue },
226.          { typeof(float[]), 1 }
227.      };
228.      public const int GROUP_ID_LENGTH = 4;
229.
230.      /// <summary>
231.      /// Checks if given file path is valid for reading and writing.
232.      /// </summary>
233.      /// <param name="sFilepath">Path to a file.</param>
234.      /// <returns>True if file path is valid, False otherwise.</returns>
235.      public static bool IsValidFilepath(string sFilepath)

```



```

236.         {
237.             // Try writing a dummy file to file path
238.             try
239.             {
240.                 if (!File.Exists(sFilepath))
241.                 {
242.                     File.WriteAllText(sFilepath, "");
243.                     File.Delete(sFilepath);
244.                 }
245.
246.                 return (true);
247.             }
248.
249.             // If file could not be written, file path is invalid
250.             catch
251.             {
252.                 return (false);
253.             }
254.         }
255.
256.         /// <summary>
257.         /// Returns index of the first occurrence of target array in source array
258.         /// ranging from the start index to search limit index. If subarray was
259.         not found,
260.         /// returns -1.
261.         /// </summary>
262.         /// <param name="arrbSource">Byte array to look for subarray in.</param>
263.         </param>
264.         /// <param name="nStartIndex">Index to begin searching from in the source
265.         e array.</param>
266.         /// <param name="nSearchLimit">Amount of bytes to scan in the source arr
267.         ay.</param>
268.         /// <returns>
269.         /// Index of the first occurrence of target array in source array.
270.         /// If subarray was not found, returns -1.
271.         /// </returns>
272.         public static int IndexOf(byte[] arrbSource,
273.                                   byte[] arrbTarget,
274.                                   int nStartIndex,
275.                                   int nSearchLimit)
276.         {
277.             // If passed values are within limits
278.             if (arrbSource.Length >= arrbTarget.Length &&
279.                 arrbTarget.Length > 0 &&
280.                 arrbSource.Length - nStartIndex > nSearchLimit &&
281.                 arrbSource.Length > nStartIndex)
282.             {
283.                 // Iterate over source array from the given index
284.                 for (int nIndexSource = nStartIndex;
285.                     nIndexSource < nStartIndex + nSearchLimit;
286.                     nIndexSource++)
287.                 {
288.                     // If the first byte of target array was found in the source
289.                     array
290.                     if (arrbSource[nIndexSource] == arrbTarget[0])
291.                     {
292.                         // Define a 'found' flag boolean as true
293.                         bool bFound = true;

```

```

290.
291.                // Iterate over target array
292.                for (int nIndexTarget = 1;
293.                    nIndexTarget < arrbTarget.Length && bFound == true;
294.                    nIndexTarget++)
295.                {
296.                    // If bytes do not match, mark 'found' flag as false
297.                    if (arrbSource[nIndexSource + nIndexTarget] != arrbT
    arget[nIndexTarget])
298.                    {
299.                        bFound = false;
300.                    }
301.                }
302.
303.                // If found flag is true after the iteration, subarray w
    as found
304.                if (bFound == true)
305.                {
306.                    return (nIndexSource);
307.                }
308.            }
309.        }
310.    }
311.
312.    // If subarray was not found, return an illegal index value
313.    return (-1);
314.}
315.}
316.
317.    /// <summary>
318.    /// Header information struct.
319.    /// </summary>
320.    public struct WAV_Header
321.    {
322.        public char[]    sGroupID;        // Chunk ID (should be 'RIFF')
323.        public uint      dwFileLength;    // File size -
    8 (without Group ID and RIFF type)
324.        public char[]    sRiffType;      // Extension of a RIFF file (should be '
    WAVE')
325.    }
326.
327.    /// <summary>
328.    /// Format chunk information struct.
329.    /// </summary>
330.    public struct WAV_FormatChunk
331.    {
332.        public char[]    sGroupID;        // Chunk ID (should be 'fmt
    ')
333.        public uint      dwChunkSize;     // Size of the rest of the c
    hunk which follows this number
334.        public ushort    wFormatTag;     // Sample format (should be
    1 for 'PCM')
335.        public ushort    wChannels;      // Amount of audio channels
    present
336.        public uint      dwSampleRate;    // Amount of samples per sec
    ond of audio
337.        public uint      dwAverageBytesPerSecond; // Average amount of bytes p
    er second audio

```

```

338.         public ushort    wBlockAlign;           // Number of audio channels
        * Bits per Sample / 8
339.         public ushort    wBitDepth;           // Amount of bits per audio
        sample
340.     }
341.
342.     /// <summary>
343.     /// Data chunk information struct.
344.     /// </summary>
345.     public struct WAV_DataChunk
346.     {
347.         public char[]    sGroupID;           // Chunk ID (should be 'data')
348.         public uint      dwChunkSize;       // Number of bytes in the sample data po
        rtion
349.         public Array     sampleData;       // Array of audio samples
350.     }
351.
352.     /// <summary>
353.     /// WAV file extension container.
354.     /// Class can decode valid WAV files, make changes to them and encode to dis
        k.
355.     /// </summary>
356.     class WAV
357.     {
358.         // Audio container information variables
359.         private static WAV_Header    hHeader;
360.         private static WAV_FormatChunk    fcFormat;
361.         private static WAV_DataChunk    dcData;
362.         private static string          sFilePath = null;
363.         private static float           fFileDuration;
364.         private static long             lFileSize;
365.         private static int              nFileBPM;
366.
367.         // Other related variables
368.         private static readonly int    BPM_WINDOW_SIZE = 8;
369.         private static readonly int    MIN_FILE_LENGTH = 36;
370.
371.         /// <summary>
372.         /// Path to loaded audio file.
373.         /// </summary>
374.         public string Path
375.         {
376.             get
377.             {
378.                 return sFilePath;
379.             }
380.         }
381.
382.         /// <summary>
383.         /// Header chunk struct of audio.
384.         /// </summary>
385.         public WAV_Header Header
386.         {
387.             get
388.             {
389.                 return hHeader;
390.             }
391.         }
392.
393.         /// <summary>
394.         /// Format chunk struct of audio.

```

```

395.         /// </summary>
396.     public WAV_FormatChunk Format
397.     {
398.         get
399.         {
400.             return fcFormat;
401.         }
402.     }
403.
404.     /// <summary>
405.     /// Data chunk struct of audio.
406.     /// </summary>
407.     public WAV_DataChunk Data
408.     {
409.         get
410.         {
411.             return dcData;
412.         }
413.     }
414.
415.     /// <summary>
416.     /// Duration of audio in seconds.
417.     /// </summary>
418.     public double Duration
419.     {
420.         get
421.         {
422.             return (IsLoaded() ? Math.Round(fFileDuration, 3) : -1);
423.         }
424.     }
425.
426.     /// <summary>
427.     /// Minimum audio length (in seconds) for BPM calculation.
428.     /// </summary>
429.     public static double BPM_CALCULATION_DURATION_MINIMUM
430.     {
431.         get
432.         {
433.             return BPM_WINDOW_SIZE * 3;
434.         }
435.     }
436.
437.     /// <summary>
438.     /// Size of audio file in bytes.
439.     /// </summary>
440.     public long Size
441.     {
442.         get
443.         {
444.             return (IsLoaded() ? lFileSize : -1);
445.         }
446.     }
447.
448.     /// <summary>
449.     /// Amount of samples per second of audio.
450.     /// </summary>
451.     public int SampleRate
452.     {
453.         get
454.         {
455.             return (IsLoaded() ? (int)fcFormat.dwSampleRate : -1);

```

```

456.         }
457.     }
458.
459.     /// <summary>
460.     /// Amount of bits per audio sample.
461.     /// </summary>
462.     public int BitDepth
463.     {
464.         get
465.         {
466.             return (IsLoaded() ? fcFormat.wBitDepth : -1);
467.         }
468.     }
469.
470.     /// <summary>
471.     /// Amount of Beats Per Minute in a given audio file.
472.     /// </summary>
473.     public int BPM
474.     {
475.         get
476.         {
477.             // Set window time to be 8 seconds
478.             int nWindow = BPM_WINDOW_SIZE;
479.
480.             // If file is loaded, BPM was not yet calculated and song length
481.             // is long enough
482.             if (IsLoaded() &&
483.                 nFileBPM == -1 &&
484.                 Duration >= nWindow * 3)
485.             {
486.                 // Calculate the amount of samples needed for 8 seconds of a
487.                 // audio
488.                 int nSamples = SampleRate * nWindow;
489.
490.                 // Create an array of window sample summations
491.                 double[] Peaks = new double[((int)Duration / 3) / nWindow];
492.
493.                 // Define a separate sample array
494.                 Array AudioSignal = null;
495.
496.                 // Copy samples to new sample array and normalize them
497.                 switch (Format.wBitDepth / 8)
498.                 {
499.                     case (sizeof(short)):
500.                         AudioSignal = Array.CreateInstance(typeof(short), Data
501.                             .sampleData.Length);
502.                         Array.Copy(Data.sampleData, AudioSignal, AudioSignal
503.                             .Length);
504.                         Samples.Normalize((short[])AudioSignal);
505.                         break;
506.                     case (sizeof(float)):
507.                         AudioSignal = Array.CreateInstance(typeof(float), Data
508.                             .sampleData.Length);
509.                         Array.Copy(Data.sampleData, AudioSignal, AudioSignal
510.                             .Length);
511.                         Samples.Normalize((float[])AudioSignal);
512.                         break;
513.                 }
514.
515.                 // Go over the middle third of the audio data

```

```

510.                for (int nWindowIndex = 0, nSampleIndex = AudioSignal.Length
                    / 3;
511.                    nWindowIndex < Peaks.Length;
512.                    nWindowIndex++)
513.                {
514.                    // Go over the current window
515.                    for (int nWindowEnd = nSampleIndex + nSamples;
516.                        nSampleIndex < nWindowEnd;
517.                        nSampleIndex++)
518.                    {
519.                        // Count the amount of 0db peaks in the current wind
ow
520.                        switch (Format.wBitDepth / 8)
521.                        {
522.                            case (sizeof(short)):
523.                                Peaks[nWindowIndex] += Math.Abs((short)Audio
Signal.GetValue(nSampleIndex) - 1) == Resources.MAX_AMPLITUDE[typeof(short[])] -
1 ? 1 : 0;
524.                                break;
525.                            case (sizeof(float)):
526.                                Peaks[nWindowIndex] += Math.Abs((float)Audio
Signal.GetValue(nSampleIndex) - 1) == Resources.MAX_AMPLITUDE[typeof(float[])] -
1 ? 1 : 0;
527.                                break;
528.                        }
529.                    }
530.                }
531.
532.                // Get the index of the first sample of the "loudest" window
533.                int nLeadSample = (Peaks.ToList().IndexOf(Peaks.Max())) * nWi
ndow) + (Peaks.Length * nWindow) * SampleRate;
534.
535.                // Define variables for Fast Fourier Transform
536.                int N = nSamples;
537.                int nPower = 0;
538.                int nIndex;
539.
540.                // Find a power of 2 that is the closest to the amount of sa
mples in a window
541.                for (nIndex = 1;
542.                    (nIndex * 2) < N;
543.                    nIndex *= 2, nPower++) ;
544.
545.                // Set the amount of samples to be 2 in power we just found
546.                N = (int)Math.Pow(2, nPower);
547.
548.                // Define arrays for real and imaginary parts of the signal
549.                double[] Real = new double[N];
550.                double[] Imaginary = new double[N];
551.
552.                // Copy the amplitude values from audio data to the real par
t of the signal
553.                switch (Format.wBitDepth / 8)
554.                {
555.                    case (sizeof(short)):
556.                        Samples.LoadSamples(Real, (short[])AudioSignal, nLea
dSample, N);
557.                        break;

```

```

558.             case (sizeof(float)):
559.                 Samples.LoadSamples(Real, (float[])AudioSignal, nLea
dSample, N);
560.                 break;
561.             }
562.
563.             // Calculate a Fast Fourier Transform over given signal
564.             Resources.FourierTransform.FFT(1, nPower, Real, Imaginary);
565.
566.             // Calculate a spectrum coefficient
567.             float nSpectrumCoefficient = (SampleRate / (float)N) * 2;
568.             int nSample;
569.
570.             // Reach the 250hz frequency bin
571.             for (nSample = 0;
572.                 nSample * nSpectrumCoefficient < 250;
573.                 nSample++) ;
574.
575.             // Set every frequency bin beginning from 100hz to be 0
576.             for (;
577.                 nSample < N;
578.                 nSample++)
579.             {
580.                 Real[nSample] = Imaginary[nSample] = 0;
581.             }
582.
583.             // Calculate an Inverse Fast Fourier Transform over given si
gnal
584.             Resources.FourierTransform.FFT(-
1, nPower, Real, Imaginary);
585.
586.             // Define variables for lowpassed signal analysis
587.             float[] Lowpass = new float[N];
588.             double dAverage = 0;
589.             double dSummation = 0;
590.
591.             // Iterate over each sample in the real array
592.             for (nIndex = 0; nIndex < N; nIndex++)
593.             {
594.                 // Copy lowpassed signal to the lowpass array, setting e
ach negative value to 0
595.                 switch (Format.wBitDepth / 8)
596.                 {
597.                     case (sizeof(short)):
598.                         Lowpass[nIndex] = Math.Max((short)0, (short)Real
[nIndex]);
599.                         break;
600.                     case (sizeof(float)):
601.                         Lowpass[nIndex] = Math.Max((float)0, (float)Real
[nIndex]);
602.                         break;
603.                 }
604.             }
605.
606.             // Normalize the received lowpass signal
607.             switch (Format.wBitDepth / 8)
608.             {
609.                 case (sizeof(short)):
610.                     // Convert the float array to short array

```

```

611.                                     Samples.Normalize(Array.ConvertAll(Lowpass, new Conv
    enter<float, short>
612.                                     (
613.                                         // Cast each float sample to short sample
614.                                         delegate (float fSample)
615.                                         {
616.                                             return (short)fSample;
617.                                         }
618.                                     ));
619.                                     break;
620.                                     case (sizeof(float)):
621.                                         Samples.Normalize(Lowpass);
622.                                         break;
623.                                     }
624.
625.                                     // Create a list of pairs for impulse matches
626.                                     List<KeyValuePair<int, int>> Matches;
627.
628.                                     // Summate lowpassed signal amplitudes
629.                                     for (nIndex = 0; nIndex < N; nIndex++)
630.                                     {
631.                                         dSummation += Lowpass[nIndex];
632.                                     }
633.
634.                                     // Set the average multiplier to start from 7
635.                                     int nMultiplier = 7;
636.
637.                                     // Start looking for bass drum impulses in the lowpassed sig
    nal.
638.                                     // Amplitude threshold is calculated by taking the average a
    mplitude value
639.                                     // and multiplying it by the multiplier variable.
640.                                     // If fewer than two matches were found above the threshold,
    lower the
641.                                     // threshold value by decrementing the multiplier value and
    try again.
642.                                     do
643.                                     {
644.                                         // Calculate average sample amplitude times the average
    multiplier
645.                                         dAverage = dSummation / N * nMultiplier--;
646.
647.                                         // Initialize a new list of matches
648.                                         Matches = new List<KeyValuePair<int, int>>();
649.
650.                                         // Calculate the jumping distance from an impulse
651.                                         int nDistance = SampleRate / 4;
652.
653.                                         // Iterate over lowpass signal
654.                                         for (nIndex = 0; nIndex < N; nIndex++)
655.                                         {
656.                                             // If an amplitude above the average value was found
657.                                             if (Lowpass[nIndex] > dAverage)
658.                                             {
659.                                                 // Find the next impulse
660.                                                 for (int nImpulse = nIndex + nDistance; nImpulse
    < N; nImpulse++)
661.                                                 {
662.                                                     // If another impulse was found
663.                                                     if (Lowpass[nImpulse] > dAverage)

```



```

664.                {
665.                    // Make a match out of the first impulse
        index and
666.                    // delta of two found impulses
667.                    Matches.Add(new KeyValuePair<int, int>(n
        Index, nImpulse - nIndex));
668.                    nImpulse += nDistance;
669.                }
670.            }
671.
672.            // Jump forward from the current impulse
673.            nIndex += nDistance;
674.        }
675.    }
676.}
677.while (Matches.Count < 2 || nMultiplier < 0);
678.
679.// If at least two matches exist
680.if (nMultiplier != 0)
681.{
682.    // Define variables for the answer pair and jump counter
683.    KeyValuePair<int, int> Answer = new KeyValuePair<int, in
        t>(0, 0);
684.    int nJumps = 0;
685.
686.    // Iterate over each impulse in the list of matches
687.    foreach (KeyValuePair<int, int> Impulse in Matches)
688.    {
689.        // Start from the second impulse index
690.        // Add delta from the pair to jump to the presumed i
        ndex of the next impulse
691.        for (nIndex = Impulse.Key + Impulse.Value; nIndex <
        N; nIndex += Impulse.Value)
692.        {
693.            // If jump landed on another impulse
694.            if (Lowpass[nIndex] > dAverage)
695.            {
696.                // Increment the jump counter
697.                nJumps++;
698.            }
699.
700.            // Otherwise fulfill the loop condition
701.            else
702.            {
703.                nIndex = N;
704.            }
705.        }
706.
707.        // If successful jumps were made
708.        if (nJumps > 0)
709.        {
710.            // Set the answer to be the current impulse delt
        a if current answer's jump amount is lower
711.            Answer = nJumps > Answer.Key ? new KeyValuePair<
        int, int>(nJumps, Impulse.Value) : Answer;
712.        }
713.
714.        // Set the jump counter back to 0;
715.        nJumps = 0;
716.    }

```

```

717.
718.                                // To calculate BPM using the sample delta, use formula:
719.                                // BPM = 60 / (Delta / Sample Rate)
720.                                nFileBPM = (int)(60 / ((float)Answer.Value / SampleRate)
721.                                );
722.                                // If BPM is lower than 100 or bigger than 200, scale it
723.                                up or down appropriately
724.                                while (nFileBPM < 100 || nFileBPM > 200)
725.                                {
726.                                    nFileBPM = nFileBPM < 100 ? nFileBPM * 2 : nFileBPM
727.                                    > 200 ? nFileBPM / 2 : nFileBPM;
728.                                }
729.                                }
730.                                return (nFileBPM);
731.                            }
732.                    }
733.
734.                    /// <summary>
735.                    /// A set of functions that work with uncompressed audio samples.
736.                    /// </summary>
737.                    class Samples
738.                    {
739.                        /// <summary>
740.                        /// Loads samples from byte array to short array.
741.                        /// </summary>
742.                        public static void LoadSamples(short[] Destination, byte[] Source, i
743.                        nt nSampleDataIndex)
744.                        {
745.                            // Calculate sample size in bytes
746.                            int nSampleSize = sizeof(short);
747.
748.                            // Load destination array with converted samples
749.                            for (int nSampleIndex = 0;
750.                                nSampleIndex < Destination.Length;
751.                                nSampleIndex++)
752.                            {
753.                                Destination[nSampleIndex] = BitConverter.ToInt16(Source, nSa
754.                                mpleDataIndex + (nSampleIndex * nSampleSize));
755.                            }
756.
757.                        /// <summary>
758.                        /// Loads samples from byte array to float array.
759.                        /// </summary>
760.                        public static void LoadSamples(float[] Destination, byte[] Source, i
761.                        nt nSampleDataIndex)
762.                        {
763.                            // Calculate sample size in bytes
764.                            int nSampleSize = sizeof(float);
765.
766.                            // Load destination array with converted samples
767.                            for (int nSampleIndex = 0;
768.                                nSampleIndex < Destination.Length;
769.                                nSampleIndex++)
770.                            {
771.                                Destination[nSampleIndex] = BitConverter.ToSingle(Source, nS
772.                                ampleDataIndex + (nSampleIndex * nSampleSize));

```

```

770.         }
771.     }
772.
773.         /// <summary>
774.         /// Loads samples from short array to double array.
775.         /// </summary>
776.         public static void LoadSamples(double[] Destination, short[] Source,
777. int nSampleDataIndex, int nSampleAmount)
778.         {
779.             // Load destination array with source samples
780.             for (int nSampleIndex = 0;
781. nSampleIndex < nSampleAmount;
782. nSampleIndex++)
783.             {
784.                 Destination[nSampleIndex] = Source[nSampleDataIndex + nSampleIndex];
785.             }
786.
787.         /// <summary>
788.         /// Loads samples from float array to double array.
789.         /// </summary>
790.         public static void LoadSamples(double[] Destination, float[] Source,
791. int nSampleDataIndex, int nSampleAmount)
792.         {
793.             // Load destination array with source samples
794.             for (int nSampleIndex = 0;
795. nSampleIndex < nSampleAmount;
796. nSampleIndex++)
797.             {
798.                 Destination[nSampleIndex] = Source[nSampleDataIndex + nSampleIndex];
799.             }
800.
801.         /// <summary>
802.         /// Normalized a signal of short samples.
803.         /// </summary>
804.         /// <param name="Samples">Signal of short samples.</param>
805.         public static void Normalize(short[] Samples)
806.         {
807.             // Define a max amplitude variable
808.             int nMaxAmplitude = 1;
809.
810.             // Iterate over sample array and find highest amplitude value
811.             for (int nSampleIndex = 0;
812. nSampleIndex < Samples.Length;
813. nSampleIndex++)
814.             {
815.                 nMaxAmplitude = Math.Abs((int)Samples[nSampleIndex]) > nMaxAmplitude ?
816.                     Math.Abs((int)Samples[nSampleIndex]) : nMaxAmplitude;
817.             }
818.
819.             // Calculate sample multiplication coefficient
820.             float fCoefficient = Resources.MAX_AMPLITUDE[typeof(short)] /
821. (float)nMaxAmplitude;
822.
823.             // If samples are not already normalized
824.             if (fCoefficient > 1)

```

```

824.         {
825.             // Iterate over sample array and multiply each sample amplit
            ude by coefficient
826.             for (int nSampleIndex = 0;
827.                 nSampleIndex < Samples.Length;
828.                 nSampleIndex++)
829.             {
830.                 Samples[nSampleIndex] = (short)(Samples[nSampleIndex] *
fCoefficient);
831.             }
832.         }
833.     }
834.
835.     /// <summary>
836.     /// Normalizes a signal of float samples.
837.     /// </summary>
838.     /// <param name="Samples">Signal of float samples.</param>
839.     public static void Normalize(float[] Samples)
840.     {
841.         // Define a max amplitude variable
842.         float fMaxAmplitude = 1;
843.
844.         // Iterate over sample array and find highest amplitude value
845.         for (int nSampleIndex = 0;
846.             nSampleIndex < Samples.Length;
847.             nSampleIndex++)
848.         {
849.             fMaxAmplitude = Math.Abs(Samples[nSampleIndex]) > fMaxAmplit
            ude ?
850.                Math.Abs(Samples[nSampleIndex]) : fMaxAmplit
            ude;
851.         }
852.
853.         // Calculate sample multiplication coefficient
854.         float fCoefficient = Resources.MAX_AMPLITUDE[(typeof(float))] /
fMaxAmplitude;
855.
856.         // If samples are not already normalized
857.         if (fCoefficient != 1)
858.         {
859.             // Iterate over sample array and multiply each sample amplit
            ude by coefficient
860.             for (int nSampleIndex = 0;
861.                 nSampleIndex < Samples.Length;
862.                 nSampleIndex++)
863.             {
864.                 Samples[nSampleIndex] *= fCoefficient;
865.             }
866.         }
867.     }
868.
869.     /// <summary>
870.     /// Reverses polarity of short signal.
871.     /// </summary>
872.     /// <param name="Samples">Short signal.</param>
873.     public static void ReversePolarity(short[] Samples)
874.     {
875.         // Iterate over sample array and multiply each amplitude by -1
876.         for (int nSampleIndex = 0;
877.             nSampleIndex < Samples.Length;
878.             nSampleIndex++)

```

```

879.         {
880.             Samples[nSampleIndex] = (short)-Samples[nSampleIndex];
881.         }
882.     }
883.
884.     /// <summary>
885.     /// Reverses polarity of float signal.
886.     /// </summary>
887.     /// <param name="Samples">Float signal.</param>
888.     public static void ReversePolarity(float[] Samples)
889.     {
890.         // Iterate over sample array and multiply each amplitude by -1
891.         for (int nSampleIndex = 0;
892.             nSampleIndex < Samples.Length;
893.             nSampleIndex++)
894.         {
895.             Samples[nSampleIndex] = -Samples[nSampleIndex];
896.         }
897.     }
898.
899.     /// <summary>
900.     /// Reverses a signal of short samples.
901.     /// </summary>
902.     /// <param name="Samples">Signal of short samples.</param>
903.     public static void Reverse(short[] Samples)
904.     {
905.         // Define a temporary sample value
906.         short nSample;
907.
908.         // Iterate over half of the sample array
909.         for (int nSampleIndex = 0;
910.             nSampleIndex < Samples.Length / 2;
911.             nSampleIndex++)
912.         {
913.             // Swap the given sample with a mirrored sample from the end
914.             nSample = Samples[Samples.Length - nSampleIndex - 1];
915.             Samples[Samples.Length - nSampleIndex -
1] = Samples[nSampleIndex];
916.             Samples[nSampleIndex] = nSample;
917.         }
918.     }
919.
920.     /// <summary>
921.     /// Reverses a signal of float samples.
922.     /// </summary>
923.     /// <param name="Samples">Signal of float samples.</param>
924.     public static void Reverse(float[] Samples)
925.     {
926.         // Define a temporary sample value
927.         float nSample;
928.
929.         // Iterate over half of the sample array
930.         for (int nSampleIndex = 0;
931.             nSampleIndex < Samples.Length / 2;
932.             nSampleIndex++)
933.         {
934.             // Swap the given sample with a mirrored sample from the end
935.             nSample = Samples[Samples.Length - nSampleIndex - 1];

```

```

936.             Samples[Samples.Length - nSampleIndex -
937. 1] = Samples[nSampleIndex];
938.             Samples[nSampleIndex] = nSample;
939.         }
940.     }
941.     /// <summary>
942.     /// Writes samples from short array to byte buffer.
943.     /// </summary>
944.     public static void WriteSamples(byte[] Buffer, short[] Samples, int
nBufferIndex)
945.     {
946.         // Calculate sample size
947.         int nSampleSize = fcFormat.wBitDepth / 8;
948.
949.         // Iterate over each sample, convert it to bytes and write to bu
ffer
950.         for (int nSampleIndex = 0;
951.             nSampleIndex < Samples.Length;
952.             nSampleIndex++)
953.         {
954.             BitConverter.GetBytes(Samples[nSampleIndex]).CopyTo(Buffer,
nBufferIndex + (nSampleIndex * nSampleSize));
955.         }
956.     }
957.
958.     /// <summary>
959.     /// Writes samples from float array to byte buffer.
960.     /// </summary>
961.     public static void WriteSamples(byte[] Buffer, float[] Samples, int
nBufferIndex)
962.     {
963.         // Calculate sample size
964.         int nSampleSize = fcFormat.wBitDepth / 8;
965.
966.         // Iterate over each sample, convert it to bytes and write to bu
ffer
967.         for (int nSampleIndex = 0;
968.             nSampleIndex < Samples.Length;
969.             nSampleIndex++)
970.         {
971.             BitConverter.GetBytes(Samples[nSampleIndex]).CopyTo(Buffer,
nBufferIndex + (nSampleIndex * nSampleSize));
972.         }
973.     }
974. }
975.
976.     /// <summary>
977.     /// Builds a WAV container for a given WAVE file.
978.     /// </summary>
979.     /// <param name="sPathToFile">Path to an audio file.</param>
980.     public WAV(string sPathToFile)
981.     {
982.         Load(sPathToFile);
983.     }
984.
985.     /// <summary>
986.     /// Builds a WAV container out of a given WAV container.
987.     /// </summary>
988.     /// <param name="waveFile">Given WAV container.</param>
989.     public WAV(WAV waveFile)

```

```

990.         {
991.             if (waveFile.IsLoaded())
992.             {
993.                 // Unload previous file information
994.                 this.Unload();
995.
996.                 // Fill in WAV header information
997.                 hHeader = new WAV_Header
998.                 {
999.                     sGroupID      = waveFile.Header.sGroupID,
1000.                     dwFileLength = waveFile.Header.dwFileLength,
1001.                     sRiffType     = waveFile.Header.sRiffType
1002.                 };
1003.
1004.                 // Fill in WAV format chunk information
1005.                 fcFormat = new WAV_FormatChunk
1006.                 {
1007.                     sGroupID      = waveFile.Format.sGroupID,
1008.                     dwChunkSize   = waveFile.Format.dwChunkSize,
1009.                     wFormatTag    = waveFile.Format.wFormatTag,
1010.                     wChannels     = waveFile.Format.wChannels,
1011.                     dwSampleRate  = waveFile.Format.dwSampleRate,
1012.                     dwAverageBytesPerSecond = waveFile.Format.dwAverageBytesPerS
econd,
1013.                     wBlockAlign  = waveFile.Format.wBlockAlign,
1014.                     wBitDepth    = waveFile.Format.wBitDepth
1015.                 };
1016.
1017.                 // Fill in WAV data chunk information
1018.                 dcData = new WAV_DataChunk
1019.                 {
1020.                     sGroupID      = waveFile.Data.sGroupID,
1021.                     dwChunkSize   = waveFile.Data.dwChunkSize
1022.                 };
1023.
1024.                 // Set file path to be a non-null value
1025.                 sFilePath = "N/A";
1026.
1027.                 // Decide upon the sample array type and load it with samples
1028.                 switch (fcFormat.wBitDepth / 8)
1029.                 {
1030.                     case (sizeof(short)):
1031.                         dcData.sampleData = Array.CreateInstance(typeof(short),
(dcData.dwChunkSize * 8) / fcFormat.wBitDepth);
1032.                         break;
1033.                     case (sizeof(float)):
1034.                         dcData.sampleData = Array.CreateInstance(typeof(float),
(dcData.dwChunkSize * 8) / fcFormat.wBitDepth);
1035.                         break;
1036.                     default:
1037.                         Unload();
1038.                         break;
1039.                 }
1040.
1041.                 // If sample data was loaded successfully
1042.                 if (IsLoaded())
1043.                 {
1044.                     // Copy audio samples to a new WAV
1045.                     waveFile.Data.sampleData.CopyTo(dcData.sampleData, 0);
1046.
1047.                     // Calculate new file size

```

```

1048.                lFileSize = System.Runtime.InteropServices.Marshal.SizeOf(hH
    eader) +          // Header size
1049.                System.Runtime.InteropServices.Marshal.SizeOf(fc
    Format) +          // Format chunk size
1050.                (sizeof(byte) * Resources.GROUP_ID_LENGTH) +
    // 'Group ID' variable type size
1051.                System.Runtime.InteropServices.Marshal.SizeOf(dc
    Data.dwChunkSize) + // 'Chunk size' variable type size
1052.                dcData.dwChunkSize;
1053.
1054.                // Set file path to be a non-null value
1055.                sFilePath = "N/A";
1056.
1057.                // WAV File Duration = Data Chunk Size / Average Bytes per S
    econd
1058.                fFileDuration = (float)dcData.dwChunkSize / fcFormat.dwAvera
    geBytesPerSecond;
1059.            }
1060.        }
1061.    }
1062.
1063.    /// <summary>
1064.    /// Builds a WAV container out of sample array, sample rate, channel amo
    unt and bit-depth.
1065.    /// </summary>
1066.    /// <param name="Data">Audio samples.</param>
1067.    /// <param name="nSampleRate">Sample rate.</param>
1068.    /// <param name="nChannels">Amount of channels.</param>
1069.    /// <param name="nBitDepth">Bit-depth.</param>
1070.    public WAV(byte[] Data,
1071.        uint nSampleRate,
1072.        ushort nChannels,
1073.        ushort nBitDepth)
1074.    {
1075.        // Unload previous file information
1076.        Unload();
1077.
1078.        // Fill in WAV header information
1079.        hHeader = new WAV_Header
1080.        {
1081.            sGroupID      = new char[] { 'R', 'I', 'F', 'F' },
1082.            dwFileLength  = 0,
1083.            sRiffType     = new char[] { 'W', 'A', 'V', 'E' }
1084.        };
1085.
1086.        // Fill in WAV format chunk information
1087.        fcFormat = new WAV_FormatChunk
1088.        {
1089.            sGroupID      = new char[] { 'f', 'm', 't', ' ' },
1090.            dwChunkSize   = 16,
1091.            wFormatTag    = 1,
1092.            wChannels     = nChannels,
1093.            dwSampleRate  = nSampleRate,
1094.            dwAverageBytesPerSecond = nSampleRate * nChannels * nBitDepth /
    8,
1095.            wBlockAlign   = (ushort)(nChannels * nBitDepth / 8),
1096.            wBitDepth     = nBitDepth
1097.        };
1098.
1099.        // Fill in WAV data chunk information
1100.        dcData = new WAV_DataChunk

```



```

1101.        {
1102.            sGroupID    = new char[] { 'd', 'a', 't', 'a' },
1103.            dwChunkSize = (uint)Data.Length
1104.        };
1105.
1106.        // Set file path to be a non-null value
1107.        sFilePath = "N/A";
1108.
1109.        // Decide upon the sample array type and load it with samples
1110.        switch (fcFormat.wBitDepth / 8)
1111.        {
1112.            case (sizeof(short)):
1113.                dcData.sampleData = Array.CreateInstance(typeof(short), (dcData.dwChunkSize * 8) / fcFormat.wBitDepth);
1114.                Samples.LoadSamples((short[])dcData.sampleData, Data, 0);
1115.                break;
1116.            case (sizeof(float)):
1117.                dcData.sampleData = Array.CreateInstance(typeof(float), (dcData.dwChunkSize * 8) / fcFormat.wBitDepth);
1118.                Samples.LoadSamples((float[])dcData.sampleData, Data, 0);
1119.                break;
1120.            default:
1121.                Unload();
1122.                break;
1123.        }
1124.
1125.        // If samples were loaded successfully
1126.        if (IsLoaded())
1127.        {
1128.            // Calculate new file size
1129.            lFileSize = System.Runtime.InteropServices.Marshal.SizeOf(hHeader) +
1130.                // Header size
1131.                System.Runtime.InteropServices.Marshal.SizeOf(fcFormat) +
1132.                // Format chunk size
1133.                (sizeof(byte) * Resources.GROUP_ID_LENGTH) +
1134.                // 'Group ID' variable type size
1135.                System.Runtime.InteropServices.Marshal.SizeOf(dcData.dwChunkSize) +
1136.                // 'Chunk size' variable type size
1137.                dcData.dwChunkSize;
1138.
1139.            // WAV File Duration = Data Chunk Size / Average Bytes per Second
1140.            fFileDuration = (float)dcData.dwChunkSize / fcFormat.dwAverageBytesPerSecond;
1141.        }
1142.    }
1143.
1144.    /// <summary>
1145.    /// Checks if given file is a WAV file.
1146.    /// </summary>
1147.    /// <param name="sPathToFile">Path to a file.</param>
1148.    /// <returns>True if given file is a valid WAV file, False otherwise.</returns>
1149.    public static bool IsWAV(string sPathToFile)
1150.    {
1151.        // If file exists and has a minimum amount of bytes
1152.        if (File.Exists(sPathToFile) &&
1153.            new FileInfo(sPathToFile).Length > MIN_FILE_LENGTH)
1154.        {
1155.            // Open a stream for a given file and read the needed amount of
1156.            bytes

```

```

1152.         FileStream fsStream = File.OpenRead(sPathToFile);
1153.         byte[] arrbBuffer = new byte[MIN_FILE_LENGTH];
1154.         fsStream.Read(arrbBuffer, 0, arrbBuffer.Length);
1155.
1156.         // Check if:
1157.         // - File is a RIFF file
1158.         // - RIFF type is WAVE
1159.         // - This codec can work with this file's bit depth
1160.         return (Encoding.ASCII.GetString(arrbBuffer, 0, 4).Equals("RIFF"
1161. ) &&
1162.         Encoding.ASCII.GetString(arrbBuffer, 8, 4).Equals("WAVE"
1163. ) &&
1164.         Resources.ALLOWED_BITDEPTHS.Contains(System.BitConverter
1165. .ToUInt16(arrbBuffer, 34)));
1166.     }
1167.
1168.     return (false);
1169. }
1170.
1171.     /// <summary>
1172.     /// Loads audio information into the given instance.
1173.     /// </summary>
1174.     /// <param name="sPathToFile">Path to an audio file (include extension).
1175.     </param>
1176.     /// <returns>True if file was loaded successfully, False otherwise.</ret
1177.     urns>
1178.     public bool Load(string sPathToFile)
1179.     {
1180.         // If file exists AND is a valid WAV file
1181.         if (File.Exists(sPathToFile) &&
1182.             IsWAV(sPathToFile))
1183.         {
1184.             // Read all bytes from a given WAV file
1185.             FileStream fs = File.OpenRead(sPathToFile);
1186.             byte[] arrbFileData = File.ReadAllBytes(sPathToFile);
1187.             int nFileDataIndex;
1188.
1189.             // Unload previous file information
1190.             Unload();
1191.
1192.             // Fill in WAV header information
1193.             hHeader = new WAV_Header
1194.             {
1195.                 sGroupID = Encoding.ASCII.GetChars(arrbFileData,
1196. (nFileDataIndex = 0), sizeof(byte) * Resources.GROUP_ID_LENGTH),
1197.                 dwFileLength = System.BitConverter.ToUInt32(arrbFileData,
1198. (nFileDataIndex += sizeof(byte) * Resources.GROUP_ID_LENGTH)),
1199.                 sRiffType = Encoding.ASCII.GetChars(arrbFileData,
1200. (nFileDataIndex += sizeof(System.Int32)), Resources.GROUP_ID_LENGTH)
1201.             };
1202.
1203.             // Find format chunk index
1204.             if ((nFileDataIndex = Resources.IndexOf(arrbFileData,
1205. Encoding.ASCII.GetBytes(
1206. "fmt "),
1207. (nFileDataIndex += sizeof
1208. f(byte) * Resources.GROUP_ID_LENGTH),
1209. System.Runtime.InteropServices.
1210. rvice.Marshal.SizeOf(fcFormat))) == -1)
1211.             {
1212.                 return (false);

```

```

1202.    }
1203.
1204.    // Fill in WAV format chunk information
1205.    fcFormat = new WAV_FormatChunk
1206.    {
1207.        sGroupID          = Encoding.ASCII.GetChars(arrbFileDa
1208.            ta,            nFileDataIndex, sizeof(byte) * Resources.GROUP_ID_LENGTH),
1209.        dwChunkSize       = System.BitConverter.ToUInt32(arrbF
1210.            ileData,      (nFileDataIndex += sizeof(byte) * Resources.GROUP_ID_LENGTH)),
1211.        wFormatTag        = System.BitConverter.ToUInt16(arrbF
1212.            ileData,      (nFileDataIndex += sizeof(System.Int32))),
1213.        wChannels         = System.BitConverter.ToUInt16(arrbF
1214.            ileData,      (nFileDataIndex += sizeof(System.Int16))),
1215.        dwSampleRate      = System.BitConverter.ToUInt32(arrbF
1216.            ileData,      (nFileDataIndex += sizeof(System.Int16))),
1217.        dwAverageBytesPerSecond = System.BitConverter.ToUInt32(arrbF
1218.            ileData,      (nFileDataIndex += sizeof(System.Int32))),
1219.        wBlockAlign       = System.BitConverter.ToUInt16(arrbF
1220.            ileData,      (nFileDataIndex += sizeof(System.Int32))),
1221.        wBitDepth         = System.BitConverter.ToUInt16(arrbF
1222.            ileData,      (nFileDataIndex += sizeof(System.Int16)))
1223.    };
1224.
1225.    // Find data chunk index
1226.    if ((nFileDataIndex = Resources.IndexOf(arrbFileData,
1227.        Encoding.ASCII.GetBytes(
1228.            "data"),
1229.            (nFileDataIndex += sizeof
1230.                f(System.Int16)),
1231.            nFileDataIndex)) == -
1232.        1)
1233.    {
1234.        return (false);
1235.    }
1236.
1237.    // Fill in WAV data chunk information
1238.    dcData = new WAV_DataChunk
1239.    {
1240.        sGroupID          = Encoding.ASCII.GetChars(arrbFileData,
1241.            nFileDataIndex, sizeof(byte) * Resources.GROUP_ID_LENGTH),
1242.        dwChunkSize       = System.BitConverter.ToUInt32(arrbFileData, (
1243.            nFileDataIndex += sizeof(byte) * Resources.GROUP_ID_LENGTH)),
1244.    };
1245.
1246.    // Shift data index to sample data start
1247.    nFileDataIndex += sizeof(System.Int32);
1248.
1249.    // Decide upon the sample array type and load it with samples
1250.    switch (fcFormat.wBitDepth / 8)
1251.    {
1252.        case (sizeof(short)):
1253.            dcData.sampleData = Array.CreateInstance(typeof(short),
1254.                (dcData.dwChunkSize * 8) / fcFormat.wBitDepth);
1255.            Samples.LoadSamples((short[])dcData.sampleData, arrbFile
1256.                Data, nFileDataIndex);
1257.            break;
1258.        case (sizeof(float)):
1259.            dcData.sampleData = Array.CreateInstance(typeof(float),
1260.                (dcData.dwChunkSize * 8) / fcFormat.wBitDepth);
1261.            Samples.LoadSamples((float[])dcData.sampleData, arrbFile
1262.                Data, nFileDataIndex);

```

```

1246.                break;
1247.            default:
1248.                Unload();
1249.                return (false);
1250.        }
1251.
1252.        // Fill in basic file information
1253.        sFilePath = sPathToFile;
1254.
1255.        // Calculate new file size
1256.        lFileSize = System.Runtime.InteropServices.Marshal.SizeOf(hHeader) +
1257.            // Header size
1258.            System.Runtime.InteropServices.Marshal.SizeOf(fcFormat) +
1259.            // Format chunk size
1260.            (sizeof(byte) * Resources.GROUP_ID_LENGTH) +
1261.            // 'Group ID' variable type size
1262.            System.Runtime.InteropServices.Marshal.SizeOf(dcData.dwChunkSize) +
1263.            // 'Chunk size' variable type size
1264.            dcData.dwChunkSize;
1265.        // Actual sample data size in bytes
1266.
1267.        // WAV File Duration = Data Chunk Size / Average Bytes per Second
1268.        fFileDuration = (float)dcData.dwChunkSize / fcFormat.dwAverageBytesPerSecond;
1269.
1270.        return (true);
1271.    }
1272.
1273.    return (false);
1274.}
1275.
1276./// <summary>
1277./// Unloads WAV instance.
1278./// </summary>
1279./// <returns>True if a file is loaded, False otherwise.</returns>
1280./// </summary>
1281./// <returns>True if a file is loaded, False otherwise.</returns>
1282./// </summary>
1283./// <summary>
1284./// Checks if a file is currently loaded within this instance.
1285./// </summary>
1286./// <returns>True if a file is loaded, False otherwise.</returns>
1287./// </summary>
1288./// <summary>
1289./// Normalizes loaded audio.
1290./// </summary>
1291./// </summary>
1292./// <summary>
1293./// Normalizes loaded audio.
1294./// </summary>
1295./// </summary>
1296./// <summary>
1297./// Normalizes loaded audio.
1298./// </summary>
1299./// </summary>

```

```

1300.         // Normalize audio based on bit depth
1301.         switch (fcFormat.wBitDepth / 8)
1302.         {
1303.             case (sizeof(short)):
1304.                 Samples.Normalize((short[])dcData.sampleData);
1305.                 break;
1306.             case (sizeof(float)):
1307.                 Samples.Normalize((float[])dcData.sampleData);
1308.                 break;
1309.         }
1310.     }
1311. }
1312.
1313.     /// <summary>
1314.     /// Reverses polarity of loaded audio.
1315.     /// </summary>
1316.     public void ReversePolarity()
1317.     {
1318.         if (IsLoaded())
1319.         {
1320.             // Reverse polarity of audio based on bit depth
1321.             switch (fcFormat.wBitDepth / 8)
1322.             {
1323.                 case (sizeof(short)):
1324.                     Samples.ReversePolarity((short[])dcData.sampleData);
1325.                     break;
1326.                 case (sizeof(float)):
1327.                     Samples.ReversePolarity((float[])dcData.sampleData);
1328.                     break;
1329.             }
1330.         }
1331.     }
1332.
1333.     /// <summary>
1334.     /// Reverses loaded audio.
1335.     /// </summary>
1336.     public void Reverse()
1337.     {
1338.         if (IsLoaded())
1339.         {
1340.             // Reverse audio based on bit depth
1341.             switch (fcFormat.wBitDepth / 8)
1342.             {
1343.                 case (sizeof(short)):
1344.                     Samples.Reverse((short[])dcData.sampleData);
1345.                     break;
1346.                 case (sizeof(float)):
1347.                     Samples.Reverse((float[])dcData.sampleData);
1348.                     break;
1349.             }
1350.         }
1351.     }
1352.
1353.     /// <summary>
1354.     /// Writes loaded audio in its current state to given destination.
1355.     /// </summary>
1356.     /// <param name="sFileDestination">Path to write audio to (include extension).</param>
1357.     /// <returns>True if file was written successfully, False otherwise.</returns>
1358.     public bool Write(string sFileDestination)

```

```

1359.         {
1360.             // If file is currently loaded AND given file path is valid
1361.             if (IsLoaded() &&
1362.                 Resources.IsValidFilepath(sFileDestination))
1363.             {
1364.                 // Create a new output stream buffer
1365.                 byte[] outputStream = new byte[lFileSize];
1366.                 int nOutputStreamIndex;
1367.
1368.                 // Write WAV header information
1369.                 Encoding.ASCII.GetBytes(hHeader.sGroupID).CopyTo(outStream,
1370.                     (nOutputStreamIndex = 0));
1371.                 BitConverter.GetBytes(lFileSize -
1372.                     8).CopyTo(outStream,
1373.                         (nOutputStreamIndex += sizeof(byte) * Resources.GROUP_ID_LENGTH));
1374.                 Encoding.ASCII.GetBytes(hHeader.sRiffType).CopyTo(outStream,
1375.                     (nOutputStreamIndex += sizeof(System.Int32)));
1376.
1377.                 // Write WAV format chunk information
1378.                 Encoding.ASCII.GetBytes(fcFormat.sGroupID).CopyTo(outStream,
1379.                     (nOutputStreamIndex += sizeof(byte) * Resources.GROUP_ID_LENGTH));
1380.                 BitConverter.GetBytes(fcFormat.dwChunkSize).CopyTo(outStream,
1381.                     (nOutputStreamIndex += sizeof(byte) * Resources.GROUP_ID_LENGTH));
1382.                 BitConverter.GetBytes(fcFormat.wFormatTag).CopyTo(outStream,
1383.                     (nOutputStreamIndex += sizeof(System.Int32)));
1384.                 BitConverter.GetBytes(fcFormat.wChannels).CopyTo(outStream,
1385.                     (nOutputStreamIndex += sizeof(System.Int16)));
1386.                 BitConverter.GetBytes(fcFormat.dwSampleRate).CopyTo(outStream,
1387.                     (nOutputStreamIndex += sizeof(System.Int16)));
1388.                 BitConverter.GetBytes(fcFormat.dwAverageBytesPerSecond).CopyTo(outStream,
1389.                     (nOutputStreamIndex += sizeof(System.Int32)));
1390.                 BitConverter.GetBytes(fcFormat.wBlockAlign).CopyTo(outStream,
1391.                     (nOutputStreamIndex += sizeof(System.Int32)));
1392.                 BitConverter.GetBytes(fcFormat.wBitDepth).CopyTo(outStream,
1393.                     (nOutputStreamIndex += sizeof(System.Int16)));
1394.
1395.                 // Write WAV data chunk information
1396.                 Encoding.ASCII.GetBytes(dcData.sGroupID).CopyTo(outStream, (nOutputStreamIndex += sizeof(System.Int16)));
1397.                 BitConverter.GetBytes(dcData.dwChunkSize).CopyTo(outStream, (nOutputStreamIndex += sizeof(byte) * Resources.GROUP_ID_LENGTH));
1398.
1399.                 // Shift stream index to sample data start
1400.                 nOutputStreamIndex += sizeof(System.Int32);
1401.
1402.                 // Write PCM samples to buffer
1403.                 switch (fcFormat.wBitDepth / 8)
1404.                 {
1405.                     case (sizeof(short)):
1406.                         Samples.WriteSamples(outStream, (short[])dcData.sampleData, nOutputStreamIndex);
1407.                         break;
1408.                     case (sizeof(float)):
1409.                         Samples.WriteSamples(outStream, (float[])dcData.sampleData, nOutputStreamIndex);
1410.                         break;
1411.                 }
1412.
1413.                 // Write byte stream to disk
1414.                 File.WriteAllBytes(sFileDestination, outputStream);
1415.
1416.             }
1417.         }

```

```

1404.             // Return true if file was written successfully
1405.             return (File.Exists(sFileDestination));
1406.         }
1407.
1408.         return (false);
1409.     }
1410.
1411.     /// <summary>
1412.     /// Returns information about the currently loaded audio.
1413.     /// </summary>
1414.     /// <returns>Returns information about the currently loaded audio.</retu
rns>
1415.     public override string ToString()
1416.     {
1417.         if (IsLoaded())
1418.         {
1419.             return ("Path: " + sFilePath + "\n" +
1420.                 "Duration: " + fFileDuration + " (s)\n" +
1421.                 "Size: " + lFileSize / (float)1000 + " (kb)\n" +
1422.                 "BPM: " + BPM + "\n" +
1423.                 "Sample Rate: " + fcFormat.dwSampleRate + " (Hz)\n" +
1424.                 "Bit Depth: " + fcFormat.wBitDepth + " (bits per sample)
\n" +
1425.                 "Channels: " + fcFormat.wChannels);
1426.         }
1427.
1428.         return ("No file loaded.");
1429.     }
1430. }
1431. }

```

2. מפרטי חומרה

ASUS ZenBook UX305 – מחשב

- CPU: 900MHz Intel Core M3-6Y30 (dual-core, 4MB cache, 2.2GHz with Turbo Boost)
- Graphics: Intel HD Graphics 515
- RAM: 8GB DDR3L (1,866 MHz SDRAM)
- Screen: 13.3-inch, 1,920 x 1,080 IPS display
- Storage: 256GB SSD (M.2 2280)
- Ports: 3 x USB 3.0, SD card reader, micro HDMI, headset jack
- Connectivity: Integrated 802.11ac
- Camera: 2MP HD webcam
- Weight: 2.6 pounds (1.18kg)
- Size: 12.8 x 8.9 x 0.48 inches (32.5 x 22.6 x 1.23cm; W x D x H)

Samsung SM-A7000 – מכשיר אנדרואיד

- CPU: Octa-core (4x1.5 GHz Cortex-A53 & 4x1.0 GHz Cortex-A53)
- Graphics: Adreno 405
- OS: Android 5.0 (Lollipop)
- RAM: 2GB

Arduino Nano – בקר ארדוינו

- Microcontroller: ATmega328
- Operating Voltage: 5V
- Input Voltage: 5-12V
- Digital I/O Pins: 14
- Analog Input Pins: 8
- Flash Memory: 32KB
- Clock Speed: 16MHz

3. מילון מונחים

DSP (Digital Signal Processing) - תחום בהנדסת חשמל, אלקטרוניקה ופיזיקה העוסק באותות, בייצוגם הספרתי ובשיטות העיבוד של אותות אלה.

Fourier Transform - כלי מרכזי באנליזה הרמונית שאפשר לתארו כפירוק של פונקציה לרכיבים מחזוריים (סינוסים וקוסינוסים או לחלופין אקספוננטים מרוכבים) וביצוע אנליזה מתמטית לפונקציה על ידי ניתוח רכיביה.

Sampling - עיבוד של אות רציף לכדי אות בדיד. דוגמה נפוצה לדגימה היא המרה של גלי קול (אות רציף בזמן) לרצף של דגימות בדידות בזמן. דגימה מייחסת ערך או קבוצת ערכים בדידים למשתנה התלוי (לרוב זמן או מרחב).

BPM - מונח מוזיקלי המגדיר את המהירות/קצב של לחן מסוים, וכן מונח המגדיר קצב פעימות הלב.

Signal - תנודה, גודל כלשהו התלוי בזמן או במרחב, או הפרעה, בעלי משמעות כלשהי.

4. מקורות

RIFF & WAVE Format

<https://blogs.msdn.microsoft.com/dawate/2009/06/23/intro-to-audio-programming-part-2-demystifying-the-wav-format/>
<http://soundfile.sapp.org/doc/WaveFormat/>
<https://en.wikipedia.org/wiki/WAV>
https://en.wikipedia.org/wiki/Resource_Interchange_File_Format

Fourier Transform

https://en.wikipedia.org/wiki/Fast_Fourier_transform
https://en.wikipedia.org/wiki/Discrete_Fourier_transform
<http://www.dspguide.com/ch12/2.htm>
<https://arxiv.org/html/math/0302212>
<http://www.dspguide.com/ch12/2.htm>

Hardware

<https://pulsesensor.com/>
<https://github.com/WorldFamousElectronics>
<https://pulsesensor.com/pages/code-and-guide>
<http://www.instructables.com/id/Arduino-AND-Bluetooth-HC-05-Connecting-easily/>

Tools

<https://www.draw.io>
<http://www.planetb.ca/syntax-highlight-word>