```
using NAudio.Wave;
using System;
using System.Data;
using System.Diagnostics;
using System.Drawing;
using System.IO;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace Karaoke
{
    public partial class Sing : Form
    {
        private WaveOut outputDevice;
        private AudioFileReader audioFile;
        private OpenFileDialog audioFileDialog;
        private OpenFileDialog lyricFileDialog;
        private DialogResult audioFileResult = DialogResult.None;
        private DialogResult lyricFileResult = DialogResult.None;
        private String audioFileName;
        private String lyricFileName;
        private Timer songTimer;
        private TimeSpan[] theTimeStamps;
        private TimeSpan[] lineLengths;
        private string[] lyricLines;
        private int totalLines;
        private int lineCount;
        private string curLine;
        private string nextLine;
        private int tickCount;
        private double amountMove;
        private TimeSpan curTime;
        private int tbp; //trackbar position
        private bool firstMove;
        private bool fromEdit;
        private bool firstTime;

        public Sing() //If the user is coming from the main screen
        {
            InitializeComponent();
            fromEdit = false;
            btnPlay.Enabled = false;
            btnPause.Enabled = false;
            btnStop.Enabled = false;
            btnForwardFifteen.Enabled = false;
            btnBackFifteen.Enabled = false;
            btnLyric.Enabled = false;
            firstTime = true;
        }

        public Sing(string audioName, string[] lines, TimeSpan[] stamps) //If the user
        is coming from the timestamp section
        {
            InitializeComponent();
            audioFileName = audioName;
            lyricLines = lines;
            theTimeStamps = stamps;
            fromEdit = true;
            firstTime = true;
            setup();
        }

        #region Btn Clicks and Playback

        private void OnPlaybackStopped(object sender, StoppedEventArgs args)
        {
            if (audioFile != null)
            {
```

```
                audioFile.Position = 0;
            }

            songTimer.Stop();
            btnPlay.Enabled = true;
            btnPause.Enabled = false;
            btnStop.Enabled = false;
            lineCount = 0;
            tickCount = 1; //How many times the timer tick has occurred, used to help move the progress bar
            curLine = "";
            nextLine = lyricLines[lineCount];
            moveLyricLine(); //Displays the current line in Pink and the next line in gray below
            curLine = nextLine;
            nextLine = lyricLines[lineCount + 1];
            amountMove = 100 / (lineLengths[0].TotalSeconds * 2); //The amount that the progress bar needs to be increased every half-second based on the length of the line
            pbLineLength.Value = 0;
            trackbarSong.Value = 0;
            curTime = TimeSpan.Zero;
            tbPosition.Text = curTime.ToString();
        }

        private void btnPlay_Click(object sender, EventArgs e)
        {
            songTimer.Start();
            if(firstTime) //So that if the user changes the audio file, a new reader will be created
            {

                outputDevice = new WaveOut();
                outputDevice.PlaybackStopped += OnPlaybackStopped;


                audioFile = new AudioFileReader(audioFileName);
                outputDevice.Init(audioFile);
                firstTime = false;

            }
            outputDevice.Play();
            btnPause.Enabled = true;
            btnPlay.Enabled = false;
            btnStop.Enabled = true;
            btnAudio.Enabled = false;
            btnLyric.Enabled = false;
            btnForwardFifteen.Enabled = true;
            btnBackFifteen.Enabled = true;
            labelFocus.Focus(); //An invisible label to shift the focus off of a button
        }

        private void btnPause_Click(object sender, EventArgs e)
        {
            outputDevice.Pause();
            songTimer.Stop();
            btnPlay.Enabled = true;
            btnPause.Enabled = false;
            labelFocus.Focus(); //An invisible label to shift the focus off of a button
        }

        private void btnStop_Click(object sender, EventArgs e)
        {
            if (audioFile != null)
            { audioFile.Position = 0; }

            outputDevice?.Stop();
            songTimer?.Stop();

```

```
            trackbarSong.Value = 0;
            tbPosition.Text = new TimeSpan(0, 0, 0).ToString();
            lineCount = 0;
            curLine = "";
            curTime = TimeSpan.Zero;
            tickCount = 1; //How many timer ticks have occurred, used to help move the progress bar
            amountMove = 100 / (lineLengths[0].TotalSeconds * 2); //How much the progress bar needs to move every half-second based on the length of the line
            pbLineLength.Value = 0;
            nextLine = lyricLines[lineCount];
            moveLyricLine(); //Change the lyrics shown to the first line as the "next" line
            curLine = nextLine;
            nextLine = lyricLines[lineCount + 1];
            btnPause.Enabled = false;
            btnStop.Enabled = false;
            btnBackFifteen.Enabled = false;
            btnForwardFifteen.Enabled = false;
            btnAudio.Enabled = true;

            firstTime = true; //Resets the firstTime boolean so that if the user changes the audio file, a new reader will be initialized

            labelFocus.Focus(); //An invisible label to shift the focus off of a button
        }

        private void btnBackFifteen_Click(object sender, EventArgs e)
        {
            btnPause.PerformClick();

            if (audioFile.CurrentTime.TotalSeconds < 15) //If we aren't 15 seconds in, just go to the beginning
            {
                audioFile.Position = 0;
            }
            else //Otherwise go backwards 15 seconds
            {
                audioFile.CurrentTime = audioFile.CurrentTime - TimeSpan.FromSeconds(15);
            }

            curTime = audioFile.CurrentTime; //Get the current time once to be used later
            tbPosition.Text = (audioFile.CurrentTime - TimeSpan.FromMilliseconds(audioFile.CurrentTime.Milliseconds) - TimeSpan.FromHours(audioFile.CurrentTime.Hours)).ToString(); //Remove the milliseconds from display
            tickCount = 0; //Reset the tick count


            tbp = (int)curTime.TotalSeconds; //Get new value for trackbar
            if (tbp < trackbarSong.Maximum)
            {
                trackbarSong.Value = tbp;
            }
            else //If greater than maximum, set to maximum
            {
                trackbarSong.Value = trackbarSong.Maximum;
            }

            findLineNumber(); //Find where we need to be in the song
            updateLines(); //Change lyrics accordingly
            labelFocus.Focus(); //An invisible label to shift the focus off of a button
        }

        private void btnForwardFifteen_Click(object sender, EventArgs e)
        {
            btnPause.PerformClick();

            if (audioFile.CurrentTime.TotalSeconds + 15 >= audioFile.TotalTime.TotalSeconds) //If there aren't 15 seconds left, just go to the end
```

```
194             {
195                 audioFile.CurrentTime = audioFile.TotalTime;
196             }
197             else //Otherwise go forward 15 seconds
198             {
199                 audioFile.CurrentTime = audioFile.CurrentTime + TimeSpan.FromSeconds(15);
200             }
201 
202             curTime = audioFile.CurrentTime; //Get the current time once to be used later
203             tbPosition.Text = (audioFile.CurrentTime - TimeSpan.FromMilliseconds(
                audioFile.CurrentTime.Milliseconds)).ToString(); //Remove the milliseconds
                for display
204             tickCount = 0; //Reset the tick count
205 
206             tbp = (int)curTime.TotalSeconds; //Get new value for trackbar
207             if (tbp < trackbarSong.Maximum)
208             {
209                 trackbarSong.Value = tbp;
210             }
211             else //If greater than maximum, then set to maximum
212             {
213                 trackbarSong.Value = trackbarSong.Maximum;
214             }
215 
216             findLineNumber(); //Find what line number we are now on
217             updateLines(); //Adjust lyrics accordingly
218             labelFocus.Focus(); //An invisible label to shift the focus off of a button
219         }
220 
221         private void btnBack_Click(object sender, EventArgs e)
222         {
223             btnStop.PerformClick(); //Stop the audio before returning to the main screen
224 
225             Main m = new Main();
226             Hide();
227             m.ShowDialog();
228         }
229 
230         private void btnAudio_Click(object sender, EventArgs e)
231         {
232             audioFileDialog = new OpenFileDialog();
233             audioFileDialog.Title = "Audio File";
234             audioFileDialog.Filter = "WAV Files (*.wav)|*.wav|M4A Files
                (*.m4a)|*.m4a|MP3 files (*.mp3)|*.mp3";
235             audioFileDialog.Title = "Audio File";
236             audioFileResult = audioFileDialog.ShowDialog();
237 
238             if (audioFileDialog.FileName.EndsWith(".mp3")) //Asks the user to convert
                MP3 files, see the documentation in the "btnAudio_Click" of Timestamp.cs for
                reasoning
239             {
240                 var filePathWav = audioFileDialog.FileName.Remove(audioFileDialog.
                    FileName.Length - 4, 4);
241                 var fileNameWav = audioFileDialog.SafeFileName.Remove(audioFileDialog.
                    SafeFileName.Length - 4, 4);
242                 if (File.Exists(filePathWav + ".wav")) //If a WAV file with same name as
                    MP3 already exists, use that
243                 {
244                     audioFileName = filePathWav + ".wav";
245                     MessageBox.Show("using \"" + fileNameWav + ".wav\" as audio file",
                        "Wav File Name");
246                     audioFileResult = DialogResult.OK;
247                     btnLyric.Enabled = true;
248                 }
249                 else //Otherwise prompt for conversion
250                 {
251                     MessageBoxButtons mb = MessageBoxButtons.OKCancel;
252                     DialogResult mbResult = MessageBox.Show("MP3 Files Must Be Converted
                        to WAV File, Click 'Okay' to continue or click 'Cancel' to choose
```

```
                    new file", "MP3 File Selected", mb);
253                 if (mbResult == DialogResult.OK)
254                 {
255                     var saveFileDialog = new SaveFileDialog();
256                     saveFileDialog.Filter = "WAV Files (*.wav)|*.wav*";
257                     saveFileDialog.Title = "Save WAV File";
258                     saveFileDialog.FileName = fileNameWav + ".wav";
259                     if (saveFileDialog.ShowDialog() == DialogResult.OK)
260                     {
261                         audioFileName = saveFileDialog.FileName;
262                         MessageBox.Show(saveFileDialog.FileName, "Wav File Name");
263                         using (MediaFoundationReader mr = new MediaFoundationReader(
                            audioFileDialog.FileName))
264                         {
265                             WaveFileWriter.CreateWaveFile(audioFileName, mr);
266                         }
267                         audioFileResult = DialogResult.OK;
268                         btnLyric.Enabled = true;
269                     }
270                 }
271
272             }
273         }
274         else if (audioFileResult == DialogResult.OK)
275         {
276             audioFileName = audioFileDialog.FileName; //Set the file name
277             btnLyric.Enabled = true;
278         }
279     }
280
281     private void btnLyric_Click(object sender, EventArgs e)
282     {
283         lyricFileDialog = new OpenFileDialog();
284         lyricFileDialog.Filter = "LRC Files (*.lrc)|*.lrc";
285         lyricFileDialog.Title = "Lyric File";
286         lyricFileResult = lyricFileDialog.ShowDialog();
287         if (lyricFileResult == DialogResult.OK)
288         {
289             lyricFileName = lyricFileDialog.FileName;
290             setup(); //Read in lyrics and timestamps
291         }
292     }
293
294     #endregion
295
296     #region Trackbar Controls
297
298     private void trackbarSetup()
299     {
300         trackbarSong.Enabled = true;
301         trackbarSong.Minimum = 0;
302         trackbarSong.Maximum = (int)audioFile.TotalTime.TotalSeconds;
303         trackbarSong.Value = 0;
304         trackbarSong.TickFrequency = 30;
305     }
306
307     private void trackbarSong_Scroll(object sender, EventArgs e)
308     {
309         int tsp = trackbarSong.Value; //Get where the trackbar is and set the audio
            file position to that value
310         audioFile.CurrentTime = TimeSpan.FromSeconds(tsp);
311     }
312
313     private void trackbarSong_MouseUp(object sender, MouseEventArgs e)
314     {
315         findLineNumber(); //Find where we need to be
316         updateLines(); //Set the values of nextLine and currentLine accordingly
317         moveLyricLine(); //Update the textbox with the correct lyrics
318         tickCount = 10; //Force the timerTick function to check values the next time
```

```
            it ticks
        }

        private void trackbarSong_MouseDown(object sender, MouseEventArgs e)
        {

        }

        #endregion

        #region Misc Functions
        private void Sing_KeyDown(object sender, KeyEventArgs e)
        {
            if (e.KeyCode == Keys.Space) //Pause or play the music
            {
                if (btnPause.Enabled)
                {
                    btnPause.PerformClick();
                }
                else if (btnPlay.Enabled)
                {
                    btnPlay.PerformClick();
                }
            }
        }

        private void setup()
        {

            if (!fromEdit) //If the user has come from the main screen
            {
                if(audioFileResult != DialogResult.OK && lyricFileResult != DialogResult.
                OK) //Make sure both audio file and lrc file have been "opened"
                {
                    return;
                }

                var allLines = File.ReadAllLines(lyricFileName);
                totalLines = allLines.Length;
                theTimeStamps = new TimeSpan[totalLines];
                lyricLines = new string[totalLines];
                lineCount = 0;
                StringBuilder sb = new StringBuilder();
                String timeStamp;
                foreach (var line in allLines) //See "editExisting" fucntion in
                Timestamp.cs for explanation of this section
                {
                    int j = 0;
                    StringBuilder time = new StringBuilder();
                    String tempString = String.Concat(line.Where(c => Char.IsDigit(c) ||
                    Char.Equals(c, '.') || Char.Equals(c, ':')));

                    var pieces = line.Split(' ');

                    timeStamp = tempString;

                    var numbers = timeStamp.Split(':');
                    var split = numbers[numbers.Length - 1].Split('.');
                    if (numbers.Length + 1 == 4)
                    {
                        theTimeStamps[lineCount] = new TimeSpan(Int32.Parse(numbers[0]),
                        Int32.Parse(numbers[1]), Int32.Parse(split[0]));
                        for (int i = 1; i < pieces.Length; i++)
                        {
                            sb.Append(pieces[i] + " ");
                        }
                    }
                    else
                    {
```

```
                        theTimeStamps[lineCount] = new TimeSpan(0, Int32.Parse(numbers[0
                        ]), Int32.Parse(split[0]));
                        pieces[0] = String.Concat(pieces[0].Where(c => Char.IsLetter(c)
                        || (Char.IsPunctuation(c) && !Char.Equals(c, ':') && !Char.Equals
                        (c, '.') && !Char.Equals(c, '[') && !Char.Equals(c, ']'))));
                        for (int i = 0; i < pieces.Length; i++)
                        {
                            sb.Append(pieces[i] + " ");
                        }

                    }

                    lyricLines[lineCount] = sb.ToString();
                    sb.Clear();
                    lineCount++;

                } //foreach

            } //!fromEdit
            else
            {
                totalLines = lyricLines.Length;
            }

            btnPlay.Enabled = true;
            btnStop.Enabled = true;
            btnPause.Enabled = false;
            btnLyric.Enabled = false;
            btnAudio.Enabled = false;

            this.FormClosing += btnStop_Click;
            this.FormClosing += btnBack_Click;

            if (outputDevice == null)
            {
                outputDevice = new WaveOut();
                outputDevice.PlaybackStopped += OnPlaybackStopped;
            }
            if (audioFile == null)
            {
                audioFile = new AudioFileReader(audioFileName);
                outputDevice.Init(audioFile);
            }

            lineLengths = new TimeSpan[totalLines];
            lineCount = 0;
            lineLengths[0] = theTimeStamps[0];
            for (int i = 1; i < totalLines - 1; i++) //Calculate how long each line of
            lyrics is
            {
                lineLengths[i] = (theTimeStamps[i] - theTimeStamps[i - 1]);
            }
            lineLengths[totalLines - 1] = (audioFile.TotalTime - theTimeStamps[totalLines
             - 1]);

            curLine = "";
            nextLine = lyricLines[lineCount];
            amountMove = 100 / (lineLengths[0].TotalSeconds * 2); //Used to move the
            progress bar every half-second

            pbLineLength.Maximum = 100;
            pbLineLength.Value = 0;

            moveLyricLine(); //update the lyrics textbox
            trackbarSetup(); //Initialize the trackbar
            curLine = nextLine;
            nextLine = lyricLines[lineCount + 1];
            tickCount = 1;
            firstMove = true;
```

```
446             curTime = TimeSpan.Zero;
447
448
449             songTimer = new Timer();
450             songTimer.Interval = 100;
451             songTimer.Tick += SongTimer_Tick;
452
453
454             tbPosition.Text = new TimeSpan().ToString();
455             labelFocus.Focus(); //An invisible label to shift the focus off of a button
456         }
457
458         private void SongTimer_Tick(object sender, EventArgs e)
459         {
460             tickCount++;
461             if (tickCount >= 5) //"half second"
462             {
463                 if (pbLineLength.Value + amountMove > pbLineLength.Maximum) //If the
                    progress bar has reached its maximum, stay at the maximum
464                 {
465                     pbLineLength.Value = pbLineLength.Maximum;
466                 }
467                 else
468                 {
469                     if (firstMove) //Move an additional "amountMove" the first time
                        since the progress bar starts at 0
470                     {
471                         pbLineLength.Value = pbLineLength.Value + (int)amountMove;
472                         firstMove = false;
473                     }
474
475                     if (pbLineLength.Value + amountMove > pbLineLength.Maximum) //If the
                        additional one didn't put it over the maximum, then make the actual
                        move
476                     {
477                         pbLineLength.Value = pbLineLength.Maximum;
478                     }
479                     else
480                     {
481                         pbLineLength.Value = pbLineLength.Value + (int)amountMove;
482                     }
483
484                 }
485                 tickCount = 1; //Reset tickCount for next move
486             }
487
488             curTime = audioFile.CurrentTime; //Get where we currently are in the song
489             tbp = (int)curTime.TotalSeconds; //Update the trackbar using that value
490
491             if (tbp < trackbarSong.Maximum)
492             {
493                 trackbarSong.Value = tbp;
494             }
495             else
496             {
497                 trackbarSong.Value = trackbarSong.Maximum;
498             }
499
500             if (lineCount < totalLines && theTimeStamps[lineCount] <= curTime) //If the
                curTime is passed where the next timeStamp is
501             {
502
503                 pbLineLength.Value = pbLineLength.Maximum;
504
505                 //Update the lyric textbox without calling a function
506                 tbLyrics.Clear();
507                 tbLyrics.SelectionFont = new Font("Tahoma", 18, FontStyle.Bold);
508                 tbLyrics.SelectionColor = System.Drawing.Color.DeepPink;
509                 tbLyrics.SelectionAlignment = HorizontalAlignment.Center;
```

```
510                 tbLyrics.AppendText(curLine);
511                 tbLyrics.SelectionColor = System.Drawing.Color.LightGray;
512                 tbLyrics.SelectionFont = new Font("Tahoma", 14, FontStyle.Bold);
513                 tbLyrics.AppendText(Environment.NewLine + Environment.NewLine +
                    Environment.NewLine + nextLine);
514                 tbLyrics.DeselectAll();
515
516
517                 lineCount++;
518                 firstMove = true; //In order to make the "additional move" the next time
519
520                 if (lineCount < totalLines - 1) //If we haven't reached the last line,
                    then get the next line and adjust necessary values
521                 {
522                     curLine = lyricLines[lineCount];
523                     nextLine = lyricLines[lineCount + 1];
524                     amountMove = 100 / (lineLengths[lineCount].TotalSeconds * 2);
525                 }
526                 else //Otherwise, the next line is empty
527                 {
528                     curLine = lyricLines[totalLines - 1];
529                     nextLine = String.Empty;
530                     amountMove = 100 / (lineLengths[totalLines - 1].TotalSeconds * 2);
531                 }
532
533                 pbLineLength.Value = 0; //Start the progress bar over
534             }
535             tbPosition.Text = (curTime - TimeSpan.FromMilliseconds(curTime.Milliseconds
                )).ToString(); //Remove the milliseconds for display
536         }
537
538         private void moveLyricLine()
539         {
540             //Updates the lyric textbox with the current line in pink and the next line
                in gray
541             tbLyrics.Clear();
542             tbLyrics.SelectionFont = new Font("Tahoma", 18, FontStyle.Bold);
543             tbLyrics.SelectionColor = System.Drawing.Color.DeepPink;
544             tbLyrics.SelectionAlignment = HorizontalAlignment.Center;
545             tbLyrics.AppendText(curLine);
546             tbLyrics.SelectionColor = System.Drawing.Color.LightGray;
547             tbLyrics.SelectionFont = new Font("Tahoma", 14, FontStyle.Bold);
548             tbLyrics.AppendText(Environment.NewLine + Environment.NewLine + Environment.
                NewLine + nextLine);
549             tbLyrics.DeselectAll();
550         }
551
552         private void findLineNumber()
553         {
554             //See "findLineNumber" in Timestamp.cs for explanation
555             int theLine = 0;
556             bool found = false;
557             TimeSpan curTime = audioFile.CurrentTime;
558             while (!found && theLine < totalLines)
559             {
560                 if (theTimeStamps[theLine] != TimeSpan.Zero)
561                 {
562                     if (theLine <= 0)
563                     {
564                         if (curTime < theTimeStamps[theLine + 1] && theTimeStamps[theLine
                             + 1] != TimeSpan.Zero)
565                         {
566                             lineCount = theLine;
567                             found = true;
568                         }
569                     }
570
571                     else if (theLine == totalLines - 1)
572                     {
```

```
573                         if (curTime > theTimeStamps[theLine - 1] && theTimeStamps[theLine
                             - 1] != TimeSpan.Zero)
574                         {
575                             lineCount = theLine;
576                             found = true;
577                         }
578                     }
579                     else
580                     {
581                         if ((curTime > theTimeStamps[theLine - 1] && curTime <
                            theTimeStamps[theLine + 1])
582                             && (theTimeStamps[theLine + 1] != TimeSpan.Zero &&
                            theTimeStamps[theLine - 1] != TimeSpan.Zero))
583                         {
584                             lineCount = theLine;
585                             found = true;
586                         }
587 
588                     }
589                     theLine++;
590                 } //if !TimeSpan.Zero
591                 else
592                 {
593                     lineCount = theLine - 1;
594                     break;
595                 }
596 
597             }//While
598 
599         }
600 
601         private void updateLines()
602         {
603             //Sets the values of curLine and nextLine based on which line we are on
604             curLine = lyricLines[lineCount];
605             if (lineCount + 1 < totalLines)
606             {
607                 nextLine = lyricLines[lineCount + 1];
608             }
609             else
610             {
611                 nextLine = "";
612             }
613         }
614 
615         #endregion
616 
617     }
618 }
619 
```