```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.IO;
using System.Linq;
using System.Runtime.InteropServices;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using NAudio.Wave;

namespace Karaoke
{
    public partial class Timestamp : Form
    {
        private WaveOut outputDevice;
        private AudioFileReader audioFile = null;
        private OpenFileDialog openFileDialog;
        private SaveFileDialog saveFileDialog;
        private String audioFileName;
        private Timer songTimer;
        private bool firstPlay = true;
        private bool allLines = false;
        private TimeSpan[] theTimeStamps;
        private TimeSpan redoTimeSpan = TimeSpan.Zero;

        private string[] lyricLines;
        private int totalLines; //The total number of timestamps that need to be
        placed/lines of lyrics
        private bool isPaused;
        private int lineCount; //What line the user is currently on

        public Timestamp()
        {
            InitializeComponent();
        }

        public Timestamp(string fName) //If the user is editing an existing
        {
            InitializeComponent();
            btnPlay.Enabled = false;
            btnPause.Enabled = false;
            btnStop.Enabled = false;
            btnUndo.Enabled = false;
            btnRedo.Enabled = false;
            trackbarSong.Enabled = false;
            btnBackFifteen.Enabled = false;
            btnForwardFifteen.Enabled = false;
            btnAudioFile.Enabled = false;

            editExisting(fName);
            songTimer = new Timer();
            songTimer.Tick += SongTimer_Tick;

            isPaused = true;
            this.FormClosing += btnStop_Click;
        }

        public Timestamp(string[] lines) //If the user is coming from the "lyric" section
        {
            InitializeComponent();
            btnPlay.Enabled = false;
            btnPause.Enabled = false;
            btnStop.Enabled = false;
            btnUndo.Enabled = false;
            btnRedo.Enabled = false;
            trackbarSong.Enabled = false;
```

```
69              btnBackFifteen.Enabled = false;
70              btnForwardFifteen.Enabled = false;
71
72              setupLyrics(lines);
73
74              songTimer = new Timer();
75              songTimer.Tick += SongTimer_Tick;
76
77              isPaused = true;
78              this.FormClosing += btnStop_Click;
79
80          }
81
82          #region BTN clicks
83          private void btnPlay_Click(object sender, EventArgs e)
84          {
85              isPaused = false;
86              labelLyrics.Focus();
87              songTimer.Start();
88              btnBackFifteen.Enabled = true;
89              btnForwardFifteen.Enabled = true;
90              btnStop.Enabled = true;
91
92
93              if (firstPlay) //Only need to initialize output device and AudioFileReader
                once when a file has been chosed
94              {
95                  if (outputDevice == null)
96                  {
97                      outputDevice = new WaveOut();
98                      outputDevice.PlaybackStopped += OnPlaybackStopped;
99                  }
100                 if (audioFile == null && openFileDialog.FileName != null)
101                 {
102                     audioFile = new AudioFileReader(audioFileName);
103
104                     outputDevice.Init(audioFile);
105                 }
106
107                 setupTrackbar(); //Set the maximum value and starting value
108                 btnAudioFile.Enabled = false; //Don't allow the user to choose a new
                    audio file unless they start over
109                 firstPlay = false;
110             }
111             outputDevice.Play();
112             btnPause.Enabled = true;
113             btnPlay.Enabled = false;
114             btnAudioFile.Enabled = false;
115         }
116
117         private void btnPause_Click(object sender, EventArgs e)
118         {
119             outputDevice.Pause();
120             isPaused = true;
121             btnPlay.Enabled = true;
122             btnPause.Enabled = false;
123         }
124
125         private void btnStop_Click(object sender, EventArgs e)
126         {
127             Reset();
128         }
129
130         private void btnFile_Click(object sender, EventArgs e)
131         {
132             tbPosition.Text = new TimeSpan(0, 0, 0).ToString();
133             openFileDialog = new OpenFileDialog();
134             openFileDialog.Title = "Audio File";
135             openFileDialog.Filter = "WAV Files (*.wav)|*.wav|M4A Files (*.m4a)|*.m4a|MP3
```

```
            files (*.mp3)|*.mp3";
136         openFileDialog.Title = "Audio File";
137         DialogResult fileResult = openFileDialog.ShowDialog();
138
139         if (openFileDialog.FileName.EndsWith(".mp3")) //MP3 Files need to be
            converted since the AudioFileReader does not support MP3s
140         {
141             var filePathWav = openFileDialog.FileName.Remove(openFileDialog.FileName.
                Length - 4, 4);
142             var fileNameWav = openFileDialog.SafeFileName.Remove(openFileDialog.
                SafeFileName.Length - 4, 4);
143             if (File.Exists(filePathWav + ".wav")) //See if there is already a wav
                file of the same name as the MP3 before converting
144             {
145                 audioFileName = filePathWav + ".wav";
146                 MessageBox.Show("using \"" + fileNameWav + ".wav\" as audio file",
                    "Wav File Name");
147                 btnPlay.Enabled = true;
148                 btnStop.Enabled = true;
149             }
150             else //Otherwise prompt the user to convert the file if they want to use
                that song
151             {
152                 MessageBoxButtons mb = MessageBoxButtons.OKCancel;
153                 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);
154                 if (mbResult == DialogResult.OK)
155                 {
156                     var saveFileDialog = new SaveFileDialog();
157                     saveFileDialog.Filter = "WAV Files (*.wav)|*.wav*";
158                     saveFileDialog.Title = "Save WAV File";
159                     saveFileDialog.FileName = fileNameWav + ".wav";
160                     if (saveFileDialog.ShowDialog() == DialogResult.OK)
161                     {
162                         audioFileName = saveFileDialog.FileName;
163                         MessageBox.Show(saveFileDialog.FileName, "Wav File Name");
164                         //Use a reader that accepts MP3s to play the song and write
                            it to a WAV file
165                         //MediaFoundationReader does not support repositioning so it
                            can't/shouldn't be used as the main audio reader
166                         using (MediaFoundationReader mr = new MediaFoundationReader(
                            openFileDialog.FileName))
167                         {
168                             WaveFileWriter.CreateWaveFile(audioFileName, mr);
169                         }
170                     }
171                     btnPlay.Enabled = true;
172                     btnStop.Enabled = true;
173                 }
174
175             }
176
177         }
178         else if (fileResult == DialogResult.OK)
179         {
180             MessageBox.Show(openFileDialog.SafeFileName, "File Name"); //Show the
                user what file is being used
181             audioFileName = openFileDialog.FileName; //Store the file name/path to
                be used later
182             btnPlay.Enabled = true;
183             btnStop.Enabled = true;
184
185         }
186     }
187
188     private void btnUndo_Click(object sender, EventArgs e)
189     {
190         btnPause.PerformClick(); //To prevent the audio from continuing to play
```

```
            songTimer.Stop();

            lineCount--; //To clear out the array
            if (lineCount < 0) //To be able to undo the final timestamp after the song has started over
            {
                lineCount = totalLines - 1;
            }

            redoTimeSpan = theTimeStamps[lineCount]; //In case we want to redo that timestamp
            theTimeStamps[lineCount] = TimeSpan.Zero;
            if (lineCount > 0)
            {
                audioFile.CurrentTime = theTimeStamps[lineCount - 1];
            }
            else
            {
                audioFile.Position = 0;
            }
            tbTimestamps.Clear();
            updateTimeTB(); //The remove the previous timestamp from the textbox
            lineCount--; //To move the colored line correctly
            TextColor(tbLyrics);
            TextColor(tbTimestamps);
            lineCount++; //To get back to the correct spot for storing/placing timestmaps
            tbPosition.Text = audioFile.CurrentTime.ToString();

            AdjustTrackbar(); //Based on updated time

            btnRedo.Enabled = true;
            btnUndo.Enabled = false;
        }

        private void btnRedo_Click(object sender, EventArgs e)
        {
            btnPause.PerformClick(); //To prevent the audio from continuing to play
            songTimer.Stop();
            btnUndo.Enabled = true;
            btnRedo.Enabled = false;

            theTimeStamps[lineCount] = redoTimeSpan; //Store the recently removed timestamp back in its spot

            audioFile.CurrentTime = theTimeStamps[lineCount]; //Adjust where in the audio we are
            updateTimeTB(); //Rewrite the timestamp in the textbox
            TextColor(tbLyrics); //Move the colored lines
            TextColor(tbTimestamps);

            tbPosition.Text = audioFile.CurrentTime.ToString();

            AdjustTrackbar();

            lineCount++; //So that we have moved on
        }

        private void btnBackFifteen_Click(object sender, EventArgs e)
        {
            if (audioFile.CurrentTime.TotalSeconds < 15) //If the song isn't 15 seconds in, just go back to the beginning
            {
                audioFile.Position = 0;
            }
            else //Otherwise just jump back 15 seconds
            {
                audioFile.CurrentTime = audioFile.CurrentTime - TimeSpan.FromSeconds(15);
            }

```

```
255             //Just removes the millisecond part of the current time to make it more
                aesthetically pleasing
256             tbPosition.Text = (audioFile.CurrentTime - TimeSpan.FromMilliseconds(
                audioFile.CurrentTime.Milliseconds) - TimeSpan.FromHours(audioFile.
                CurrentTime.Hours)).ToString();
257
258             AdjustTrackbar();
259
260             findLineNumber(); //Figure out where the lyrics now need to be
261             labelLyrics.Focus(); //To remove the focus off of a button so that when
                'enter' is pressed it doesn't trigger the button
262         }
263
264         private void btnForwardFifteen_Click(object sender, EventArgs e)
265         {
266             if (audioFile.CurrentTime.TotalSeconds + 15 >= audioFile.TotalTime.
                TotalSeconds) //If there aren't 15 seconds left, just jump to the end
267             {
268                 audioFile.CurrentTime = audioFile.TotalTime;
269             }
270             else //otherwise jump forward 15 seconds
271             {
272                 audioFile.CurrentTime = audioFile.CurrentTime + TimeSpan.FromSeconds(15);
273             }
274
275             //Remove the millisecond part from the current time
276             tbPosition.Text = (audioFile.CurrentTime - TimeSpan.FromMilliseconds(
                audioFile.CurrentTime.Milliseconds)).ToString();
277
278             AdjustTrackbar();
279
280             findLineNumber(); //Figure out where the lyrics need to be
281             labelLyrics.Focus(); //Remove focus on button so that pressing 'enter'
                doesn't trigger button click
282         }
283
284         private void btnExport_Click(object sender, EventArgs e)
285         {
286             btnPause.PerformClick();
287             if (allLines) //If the user has placed all of the necessary timestamps
288             {
289                 string file = "";
290                 StringBuilder sb = new StringBuilder();
291                 saveFileDialog = new SaveFileDialog();
292                 saveFileDialog.Filter = "LRC Files (*.lrc)|*.lrc";
293                 if (saveFileDialog.ShowDialog() == DialogResult.OK)
294                 {
295                     file = saveFileDialog.FileName;
296                     for (int i = 0; i < totalLines; i++) //Write timestamps followed by
                        lyrics to file in "[timestamp] lyrics" format
297                     {
298                         sb.Append("[" + theTimeStamps[i].ToString() + "] " + lyricLines[i
                            ] + Environment.NewLine);
299                     }
300
301                     File.WriteAllText(file, sb.ToString());
302                     MessageBox.Show("File Successfully Exported"); //Let the user know
                        that it worked
303                 }
304
305             }
306             else
307             {
308                 //A user must place all timestamps before they can export
309                 MessageBox.Show($"You have not finished timestamping for this song,
                    please finish all {totalLines} lines before exporting");
310             }
311         }
312
```

```
313         private void btnSaveandContinue_Click(object sender, EventArgs e)
314         {
315             if(audioFile == null) //In case the user came from the "edit" button rather
                than lyric section
316             {
317                 MessageBox.Show("Please choose an audio file before continuing",
                    "Invalid");
318             }
319             else if (allLines) //Make sure all timestamps have been placed before
                continuing
320             {
321                 Sing s = new Sing(audioFileName, lyricLines, theTimeStamps);
322                 Hide();
323                 s.ShowDialog();
324             }
325             else
326             {
327                 //Notify if the user hasn't placed all timestamps
328                 MessageBox.Show("You must place all TimeStamps before continuing",
                    "Invalid");
329             }
330         }
331 
332         private void btnHelp_Click(object sender, EventArgs e)
333         {
334             //Displays the instructions for placing timestamps
335             String instructions = "Choose your audio file and press play to begin" +
336                 "\n\nPlace timestamps at the beginning of each line of lyrics by
                    pressing the \"Enter\" key" +
337                 "\n\nPress the \"Spacebar\" to play or pause the music" +
338                 "\n\nPress the undo button or \"CTRL + Z\" to remove the most recently
                    placed timestamp" +
339                 "\n\nPress the redo button or \"CTRL + Y\" to add an undone timestamp
                    back" +
340                 "\n\nPress the \"Export\" button once you've placed all timestamps to
                    save your LRC file" +
341                 "\n\nPress the \"Save and Continue\" button to sing along to your
                    karaoke lyrics";
342             MessageBox.Show(instructions, "How To");
343         }
344 
345         private void btnBack_Click(object sender, EventArgs e)
346         {
347             //Goes back to the main screen
348             Main m = new Main();
349             Hide();
350             m.ShowDialog();
351         }
352 
353         #endregion //btn clicks
354 
355         #region Trackbar Controls
356 
357         private void trackbarSong_Scroll(object sender, EventArgs e)
358         {
359             int ms = trackbarSong.Value; //Get where the user has scrolled to
360             audioFile.CurrentTime = TimeSpan.FromSeconds(ms); //Update the position of
                the song to where they scrolled to
361 
362         }
363 
364         private void trackbarSong_MouseUp(object sender, MouseEventArgs e)
365         {
366             //Once the user has finished scrolling
367 
368             findLineNumber(); //Determine where in the lyrics we need to be
369             TextColor(tbLyrics); //Update the color of both textboxes based on where we
                are in the song
370             TextColor(tbTimestamps);
```

```
371             if (songTimer != null) //Resume the timer
372             {
373                 songTimer.Start();
374             }
375         }
376 
377         private void trackbarSong_MouseDown(object sender, MouseEventArgs e)
378         {
379             btnPause.PerformClick(); //Pause the music
380             if (songTimer != null) //Pause the timer
381             {
382                 songTimer.Stop();
383             }
384         }
385 
386         public void setupTrackbar()
387         {
388             trackbarSong.Enabled = true;
389             trackbarSong.Minimum = 0;
390             trackbarSong.Maximum = (int)audioFile.TotalTime.TotalSeconds; //How many
                seconds the song is
391             trackbarSong.Value = 0; //Start at the beginning
392             trackbarSong.TickFrequency = 15;
393         }
394 
395         public void AdjustTrackbar()
396         {
397             //Adjusts where the tracker is on the trackbar is after reposistioning of
                the audio has occurred
398 
399             int ms = (int)audioFile.CurrentTime.TotalSeconds; //Get where we currently
                are in the song
400 
401             if (ms > trackbarSong.Maximum) //If it gets above the maximum somehow, just
                set the trackbar value to the maximum
402             {
403 
404                 trackbarSong.Value = trackbarSong.Maximum;
405             }
406             else
407             {
408                 trackbarSong.Value = ms; //Otherwise set to where we currently are
409             }
410         }
411 
412 
413         #endregion
414 
415         #region Misc Functions
416 
417         private void SongTimer_Tick(object sender, EventArgs e)
418         {
419             TimeSpan curTime = audioFile.CurrentTime;
420             int ms = (int)curTime.TotalSeconds; //Get where we currently are in the song
421 
422             if (ms > trackbarSong.Maximum) //If greater than the maximum value of the
                trackbar, set the value of trackbar to maximum
423             {
424 
425                 trackbarSong.Value = trackbarSong.Maximum;
426             }
427             else //Otherwise set the value of the trackbar to the current position in
                the song
428             {
429                 trackbarSong.Value = ms;
430             }
431 
432             if (tbTimestamps.Lines.Length >= totalLines) //Has the user placed all of
                the neccessary timestamps
```

```
433             {
434                 allLines = true;
435             }
436             else
437             {
438                 allLines = false;
439             }
440 
441             //If the current position of the song has gone past an already placed
                timestamp
442             if (lineCount < totalLines && (theTimeStamps[lineCount] <= curTime &&
                theTimeStamps[lineCount] != TimeSpan.Zero))
443             {
444                 TextColor(tbTimestamps); //Update the colored line of both textboxes
445                 TextColor(tbLyrics);
446                 lineCount++; //Move to next line
447             }
448 
449             tbPosition.Text = (curTime - TimeSpan.FromMilliseconds(curTime.Milliseconds
                )).ToString(); //Remove milliseconds for display
450 
451 
452             if (tbTimestamps.Lines.Length > 1) //If there is at least one timestamp to
                undo
453             {
454                 btnUndo.Enabled = true;
455             }
456             else
457             {
458                 btnUndo.Enabled = false;
459             }
460 
461             if (redoTimeSpan != TimeSpan.Zero) //If the user has undone a timestamp,
                they can then redo
462             {
463                 btnRedo.Enabled = true;
464             }
465             else
466             {
467                 btnRedo.Enabled = false;
468             }
469         }
470 
471         private void OnPlaybackStopped(object sender, StoppedEventArgs args)
472         {
473             if (audioFile != null)
474             {
475                 audioFile.Position = 0; //Start the audio file over
476             }
477             //Enable and disable buttons accordingly
478             btnPlay.Enabled = true;
479             btnPause.Enabled = false;
480             btnStop.Enabled = false;
481             btnBackFifteen.Enabled = false;
482             btnForwardFifteen.Enabled = false;
483             btnAudioFile.Enabled = true;
484 
485             TextColor(tbLyrics); //Change the color of lines as necessary
486             TextColor(tbTimestamps);
487             if (tbTimestamps.Lines.Length >= totalLines) //Has the user placed all of
                the neccessary timestamps
488             {
489                 allLines = true;
490                 lineCount = 0; //Start back at beginning to go through song again if
                    desired
491             }
492         }
493 
494         private void Timestamp_KeyDown(object sender, KeyEventArgs e)
```

```
495         {
496             if (e.KeyCode == Keys.Space) //Play or pause the music
497             {
498                 if (btnPause.Enabled)
499                 {
500                     btnPause.PerformClick();
501                 }
502                 else if (btnPlay.Enabled)
503                 {
504                     btnPlay.PerformClick();
505                 }
506             }
507 
508             else if (e.KeyCode == Keys.Enter && !isPaused) //Place a timestamp as long
                as music is playing
509             {
510                 if (redoTimeSpan != TimeSpan.Zero)
511                 {
512                     redoTimeSpan = TimeSpan.Zero; //Clear out redo if necessary as soon
                        as new timestamp is placed
513                 }
514 
515                 if (lineCount < totalLines) //If the user hasn't placed all timestamps
                    already
516                 {
517                     if (lineCount < 0)
518                     {
519                         lineCount = 0;
520                     }
521 
522                     theTimeStamps[lineCount] = audioFile.CurrentTime; //Store the
                        current position of audio file
523 
524                     TextColor(tbLyrics); //Update the colored line
525                     updateTimeTB(); //Update timestamp textbox to show new timestamp
526                     TextColor(tbTimestamps); //Change the color
527 
528                     lineCount++; //Move to next line for next time
529 
530                 }
531                 else //If the user has placed all timestamps, let them know they can't
                    place anymore
532                 {
533                     btnPause.PerformClick();
534                     MessageBox.Show($"You have already placed all {totalLines}
                        timestamps");
535                 }
536             }
537 
538             else if (e.Control && e.KeyCode == Keys.Z) //Undo keyboard shortcut
539             {
540 
541                 btnUndo.PerformClick();
542             }
543 
544             else if (e.Control && e.KeyCode == Keys.Y) //Redo keyboard shortcut
545             {
546                 btnRedo.PerformClick();
547             }
548         }
549 
550         private void findLineNumber()
551         {
552             int theLine = 0;
553             bool found = false;
554             TimeSpan curTime = audioFile.CurrentTime;
555             while (!found && theLine < totalLines)
556             {
557                 if (theTimeStamps[theLine] != TimeSpan.Zero) //Make sure that the
```

```
                current line has actually been placed
558             {
559                 if (theLine <= 0) //So that we don't look at a spot in the array
                    that doesn't exist
560                 {
561                     if (curTime < theTimeStamps[theLine + 1] && theTimeStamps[theLine
                        + 1] != TimeSpan.Zero)
562                     {
563                         lineCount = theLine;
564                         found = true;
565                     }
566                 }
567 
568                 else if (theLine == totalLines - 1) //So that we don't look at a
                    spot in the array that doesn't exist
569                 {
570                     if (curTime > theTimeStamps[theLine - 1] && theTimeStamps[theLine
                        - 1] != TimeSpan.Zero)
571                     {
572                         lineCount = theLine;
573                         found = true;
574                     }
575                 }
576                 else
577                 {
578                     //See if the current time belongs in the current "line" (i.e.
                    between the previous and next lines)
579                     if ((curTime > theTimeStamps[theLine - 1] && curTime <
                    theTimeStamps[theLine + 1])
580                         && (theTimeStamps[theLine + 1] != TimeSpan.Zero &&
                        theTimeStamps[theLine - 1] != TimeSpan.Zero))
581                     {
582                         lineCount = theLine;
583                         found = true;
584                     }
585 
586                 }
587                 theLine++;
588             } //if !TimeSpan.Zero
589             else
590             {
591                 lineCount = theLine - 1; //Set the linecount to the last line
592                 break;
593             }
594 
595         }//While
596     }
597 
598     private void setupLyrics(string[] lyrics)
599     {
600         btnHelp.PerformClick();
601         tbLyrics.Clear();
602         lyricLines = new string[lyrics.Length]; //Create an array to hold the lyrics
603         lineCount = 0;
604         StringBuilder sb = new StringBuilder();
605         foreach(string line in lyrics)
606         {
607             sb.Append(lineCount + 1 + ": " + line + Environment.NewLine); //Show the
                lyrics with the corresponding line number in the textbox
608             tbLyrics.AppendText(sb.ToString());
609             sb.Clear();
610             lyricLines[lineCount] = line;
611             lineCount++;
612         }
613         tbLyrics.ReadOnly = true; //Don't let the user change anything in the textbox
614         totalLines = lyricLines.Length;
615         theTimeStamps = new TimeSpan[totalLines];
616         for(int i = 0; i < totalLines; i++) //Initialize all of the timestamps as
            TimeSpan.Zero
```

```
617             {
618                 theTimeStamps[i] = TimeSpan.Zero;
619             }
620             allLines = false;
621             lineCount = 0; //Start at the very beginning
622         }
623 
624         private void editExisting(string fileName)
625         {
626             btnHelp.PerformClick();
627             var theLines = File.ReadAllLines(fileName); //Read in everything from the
                file
628             totalLines = theLines.Length;
629             theTimeStamps = new TimeSpan[totalLines];
630             lyricLines = new string[totalLines];
631             StringBuilder sb = new StringBuilder();
632             lineCount = 0;
633             foreach (string line in theLines)
634             {
635                 int j = 0;
636                 StringBuilder time = new StringBuilder();
637 
638                 String timeStamp = String.Concat(line.Where(c => Char.IsDigit(c) || Char.
                    Equals(c, '.') || Char.Equals(c, ':'))); //Remove the timestamp part
                    from the line
639 
640                 var pieces = line.Split(' '); //Split the line into pieces so that we
                    can skip over timestamp
641 
642                 var numbers = timeStamp.Split(':'); //Split the timestamp into pieces
643                 var split = numbers[numbers.Length - 1].Split('.'); //Split the last
                    piece of the timestamp into seconds and milliseconds
644                 if (numbers.Length + 1 == 4) //If the LRC file was created using this
                    program, the TimeSpan element in the file is different and contains the
                    'hour' value as well as minutes, seconds, and milliseconds
645                 {
646                     theTimeStamps[lineCount] = new TimeSpan(Int32.Parse(numbers[0]),
                        Int32.Parse(numbers[1]), Int32.Parse(split[0])); //Create the
                        timestamp
647                     for (int i = 1; i < pieces.Length; i++) //Skip over the timestamp
                        but reconnect all other pieces of the line
648                     {
649                         sb.Append(pieces[i] + " ");
650                     }
651                 }
652                 else //LRC files downloaded from the internet just have minute, second,
                    and millisecond
653                 {
654                     //Everything else is same as above
655                     theTimeStamps[lineCount] = new TimeSpan(0, Int32.Parse(numbers[0]),
                        Int32.Parse(split[0]));
656                     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, ']'))));
657                     for (int i = 0; i < pieces.Length; i++)
658                     {
659                         sb.Append(pieces[i] + " ");
660                     }
661 
662                 }
663 
664                 lyricLines[lineCount] = sb.ToString(); //Store the line
665                 sb.Clear(); //Clear the stringbuilder for the next line
666                 lineCount++;
667             } //Foreach
668 
669             lineCount = 0;
670             //Setup the textboxes as necessary
671             for (int i = 0; i < totalLines; i++)
```

```
672             {
673                 tbLyrics.AppendText(i + 1 + ": " + lyricLines[i] + Environment.NewLine);
674             }
675             for (int i = 0; i < totalLines; i++)
676             {
677                 tbTimestamps.AppendText(i + 1 + ": [" + theTimeStamps[i] + "]" +
                    Environment.NewLine);
678             }
679             //Color the first lines
680             TextColor(tbLyrics);
681             TextColor(tbTimestamps);
682             allLines = true;
683             btnAudioFile.Enabled = true;
684         }
685 
686         private void updateTimeTB()
687         {
688             tbTimestamps.Clear(); //Clear it all out
689 
690             StringBuilder sb = new StringBuilder();
691 
692             for (int i = 0; i < totalLines; i++) //Go through all lines
693             {
694                 if (theTimeStamps[i] != TimeSpan.Zero) //Only write lines that have a
                    value not equal to TimeSpan.Zero
695                 {
696                     sb.Append(i + 1 + ": [" + theTimeStamps[i] + "]" + Environment.
                        NewLine);
697                 }
698             }
699             tbTimestamps.Text = sb.ToString();
700         }
701 
702         private void TextColor(RichTextBox tb)
703         {
704             string[] lines = tb.Lines; //Get the text that is being updated
705             int vertPos = ScrollAPIs.GetScrollPosition(tb.Handle, ScrollAPIs.
                ScrollbarDirection.Vertical); //Maintain the scrollbar's position
706             tb.Clear();
707             tb.SuspendLayout();
708             for (int i = 0; i < lines.Length; i++) //Go through all of the lines
709             {
710                 if (i == lineCount) //If we're at the one that needs to be colored, then
                    use DeepPink
711                 {
712                     tb.SelectionColor = System.Drawing.Color.DeepPink;
713 
714                 }
715                 else //Otherwise use Black
716                 {
717                     tb.SelectionColor = System.Drawing.Color.Black;
718                 }
719 
720                 tb.AppendText(lines[i]);
721                 if (i != lines.Length - 1) //Don't add a NewLine after the last line
722                 {
723                     tb.AppendText(Environment.NewLine);
724                 }
725 
726             }
727 
728             if (tb == tbTimestamps && !allLines) //If not all lines have been placed,
                scroll to the bottom of the timestamp textbox
729             {
730                 tb.ScrollToCaret();
731             }
732             else //Otherwise just scroll to where we were before
733             {
734                 tb.SelectionStart = 0;
```

```
735                 tb.SelectionLength = 0;
736                 ScrollAPIs.SetScrollPosition(tb.Handle, ScrollAPIs.ScrollbarDirection.
                    Vertical, vertPos);
737             }
738             tb.ResumeLayout();
739             labelLyrics.Focus();
740         }
741 
742         private void Reset()
743         {
744             if (audioFile != null)
745             { audioFile.Position = 0; } //Start at the beginning
746 
747             outputDevice?.Stop();
748             songTimer?.Stop();
749 
750 
751             trackbarSong.Value = 0; //Reset the trackbar
752             tbPosition.Text = new TimeSpan(0, 0, 0).ToString();
753             tbTimestamps.Text = "";
754             audioFile = null; //To reset and reload the audioplayer
755 
756             for (int i = 0; i < theTimeStamps.Length; i++) //Remove all timestamps
757             {
758                 theTimeStamps[i] = TimeSpan.Zero;
759             }
760 
761             lineCount = 0;
762 
763             firstPlay = true;
764             isPaused = true;
765 
766             btnPause.Enabled = false;
767             btnPlay.Enabled = true;
768             btnAudioFile.Enabled = true;
769         }
770 
771         private void Timestamp_Load(object sender, EventArgs e)
772         {
773             btnHelp.PerformClick(); //Show instructions on load
774         }
775 
776         #endregion
777 
778         #region Scroll APIs From https:
            //stackoverflow.com/questions/21894017/textbox-maintain-scroll-bar-position-durin
            g-text-updates
779 
780         //Allows me to maintain the position of the scrollbar as I update the textbox
781         public static class ScrollAPIs
782         {
783             [DllImport("user32.dll")]
784             internal static extern int GetScrollPos(IntPtr hWnd, int nBar);
785 
786             [DllImport("user32.dll")]
787             internal static extern int SetScrollPos(IntPtr hWnd, int nBar, int nPos, bool
                 bRedraw);
788 
789             [DllImport("user32.dll")]
790             internal static extern int SendMessage(IntPtr hWnd, int wMsg, IntPtr wParam,
                IntPtr lParam);
791 
792             public enum ScrollbarDirection
793             {
794                 Horizontal = 0,
795                 Vertical = 1,
796             }
797 
798             private enum Messages
```

```
799             {
800                 WM_HSCROLL = 0x0114,
801                 WM_VSCROLL = 0x0115
802             }
803 
804             public static int GetScrollPosition(IntPtr hWnd, ScrollbarDirection direction
                )
805             {
806                 return GetScrollPos(hWnd, (int)direction);
807             }
808 
809             public static void GetScrollPosition(IntPtr hWnd, out int horizontalPosition,
                 out int verticalPosition)
810             {
811                 horizontalPosition = GetScrollPos(hWnd, (int)ScrollbarDirection.
                    Horizontal);
812                 verticalPosition = GetScrollPos(hWnd, (int)ScrollbarDirection.Vertical);
813             }
814 
815             public static void SetScrollPosition(IntPtr hwnd, int hozizontalPosition, int
                 verticalPosition)
816             {
817                 SetScrollPosition(hwnd, ScrollbarDirection.Horizontal, hozizontalPosition
                    );
818                 SetScrollPosition(hwnd, ScrollbarDirection.Vertical, verticalPosition);
819             }
820 
821             public static void SetScrollPosition(IntPtr hwnd, ScrollbarDirection
                direction, int position)
822             {
823                 //move the scroll bar
824                 SetScrollPos(hwnd, (int)direction, position, true);
825 
826                 //convert the position to the windows message equivalent
827                 IntPtr msgPosition = new IntPtr((position << 16) + 4);
828                 Messages msg = (direction == ScrollbarDirection.Horizontal) ? Messages.
                    WM_HSCROLL : Messages.WM_VSCROLL;
829                 SendMessage(hwnd, (int)msg, msgPosition, IntPtr.Zero);
830             }
831         } //End of ScrollAPIs from StackOverflow
832 
833         #endregion
834 
835     }
836 }
837 
```