**Audio extractor backup**

**This is a GUI for ffmpeg. It's value is to avoid needing to remember specific commands and flags for a cli.**

**It accepts an MKV as input, and then allows the user to select the audio streams to extract either in wav or flac formats. The user can select either lossless or lossy decoding, and select to output either as multichannel file or seperate, single channel files.**

**The program can import an mkv with multiple audio streams, including dts:x or dolby atmos encoded streams and then extract whatever streams are desired.**

**It requires ffmpeg to be installed on the host machine, and for the environmental variables setting in windows to include the path to ffmpeg.exe.**

**Intrusctionst to Add ffmpeg.exe to system PATH**

1. **Hit Start, search for Environment Variables**
2. **Click Edit the system environment variables**
3. **Under System variables, find Path → Edit**
4. **Add the folder path where ffmpeg.exe lives**

Form1.cs file requirements:

* Reads all audio streams and channel layouts dynamically using ffprobe
* Allows selecting a stream and extracting channels to separate mono files (named per channel role like FL, FR, etc.) or to a single multichannel file
* Supports both WAV and FLAC output
* Provides progress updates and estimated remaining time
* Uses proper verbatim interpolated strings for paths

Methods:

1. Browse" button click event, which will allow the user to select an MKV file, read its audio streams with ffprobe, and display the audio stream options in the list box.
   * **BtnBrowse\_Click**: This is the event handler for the "Browse" button. It opens an OpenFileDialog for selecting an MKV file, then sets the txtFilePath and calls LoadAudioStreams() to populate the list.
   * **LoadAudioStreams**: This method checks if a file is selected, clears the previous list of audio streams, runs ffprobe via RunFFProbe(), and parses the resulting JSON output to extract audio stream details.
   * **RunFFProbe**: This method runs ffprobe with arguments to gather stream information about the selected MKV file and returns the output in JSON format.
   * **ParseAudioStreams**: This method parses the JSON output from ffprobe to extract relevant information about the audio streams (codec, channels, channel layout) and stores them in a list of AudioStream objects.
   * **AudioStream**: A class to represent individual audio streams. The ToString method formats how each audio stream will be displayed in the list.
2. Extract Audio. When the "Extract Audio" button is clicked, we will extract the selected audio stream to the chosen output format (WAV or FLAC) and channel layout (multichannel or separate mono files).
   * BtnExtract\_Click: When the "Extract Audio" button is clicked, this method first checks if an audio stream is selected from the list. It then sets the output format and channel layout options based on the user's choice (WAV or FLAC, multichannel or separate mono files), and finally calls the ExtractAudio() method to begin the extraction.
   * ExtractAudio: This method constructs the FFmpeg command for audio extraction. It handles both multichannel (as a single file) and separate mono file extraction. It uses ffmpeg to extract the audio stream, either as a single file or multiple mono files. The output directory is created if it doesn't exist.
   * If separateMonoFiles is true, it maps each audio channel to a separate mono WAV file. If it's false, it extracts the entire multichannel audio as a single file.
   * lstAudioStreams\_SelectedIndexChanged: This event is triggered when the user selects an audio stream from the list. It updates the selectedAudioStream and displays the selected stream’s information in the extractionStatusLabel.
3. Progress Tracking. Updated ExtractAudio Method
4. OutputDataReceived:
   * We listen to the FFmpeg output to capture any progress data. Specifically, FFmpeg will provide time information (time=) as part of its output. This time data represents the amount of audio that has been processed so far.
     + The time= value is parsed to extract hours, minutes, and seconds, which are then used to calculate the percentage of completion.
     + We update the progress bar and the remaining time label using this data.
   * Progress Calculation:
     + The processTotalDuration variable represents the total duration of the audio stream. You'll need to calculate or parse this duration before starting the extraction process. This could be done using FFprobe or by retrieving the stream's total length from the MKV file.
     + extractionProgressBar.Value is updated based on the percentage of time elapsed from the total duration.
     + The remainingTimeLabel is updated to show the estimated remaining time of the extraction process.
   * **Calculate Total Duration**: You would need to parse the total duration of the audio stream before starting the extraction process. This could be done using FFprobe to analyze the stream and determine its length, which would help in calculating the percentage of completion.

Semi working

using System;

using System.Diagnostics;

using System.IO;

using System.Text.Json;

using System.Text.RegularExpressions;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace AudioExtractor

{

public partial class Form1 : Form

{

public Form1()

{

InitializeComponent();

}

private string GetSelectedStream()

{

if (lstAudioStreams.SelectedItem == null)

{

MessageBox.Show("Please select an audio stream to extract.", "No Stream Selected", MessageBoxButtons.OK, MessageBoxIcon.Warning);

return null;

}

string selected = lstAudioStreams.SelectedItem.ToString();

var match = Regex.Match(selected, @"Stream (\d+)");

return match.Success ? match.Groups[1].Value : null;

}

private async void BtnBrowse\_Click(object sender, EventArgs e)

{

using OpenFileDialog openFileDialog = new OpenFileDialog();

openFileDialog.Filter = "MKV files (\*.mkv)|\*.mkv";

if (openFileDialog.ShowDialog() == DialogResult.OK)

{

txtFilePath.Text = openFileDialog.FileName;

lstAudioStreams.Items.Clear();

await LoadAudioStreams(openFileDialog.FileName);

}

}

private async Task LoadAudioStreams(string filePath)

{

string args = $"-v quiet -print\_format json -show\_streams -select\_streams a \"{filePath}\"";

ProcessStartInfo psi = new ProcessStartInfo

{

FileName = "ffprobe",

Arguments = args,

RedirectStandardOutput = true,

UseShellExecute = false,

CreateNoWindow = true

};

using Process proc = new Process { StartInfo = psi };

proc.Start();

string output = await proc.StandardOutput.ReadToEndAsync();

proc.WaitForExit();

using JsonDocument doc = JsonDocument.Parse(output);

var streams = doc.RootElement.GetProperty("streams");

int index = 0;

foreach (var stream in streams.EnumerateArray())

{

string codec = stream.GetProperty("codec\_name").GetString();

int channels = stream.GetProperty("channels").GetInt32();

string channelLayout = stream.TryGetProperty("channel\_layout", out var layout) ? layout.GetString() : "unknown";

lstAudioStreams.Items.Add($"Stream {index}: {codec}, {channels}ch, {channelLayout}");

index++;

}

}

private async void BtnExtract\_Click(object sender, EventArgs e)

{

string filePath = txtFilePath.Text;

string streamIndex = GetSelectedStream();

if (filePath == "" || streamIndex == null)

return;

string extension = rbWav.Checked ? "wav" : "flac";

bool splitMono = rbMonoFiles.Checked;

string outputDir = Path.Combine(Path.GetDirectoryName(filePath), "ExtractedAudio");

Directory.CreateDirectory(outputDir);

extractionStatusLabel.Text = "Starting extraction...";

extractionProgressBar.Value = 0;

remainingTimeLabel.Text = "";

// Get duration

double duration = 0;

{

string argsProbe = $"-v quiet -print\_format json -show\_streams -select\_streams a:{streamIndex} \"{filePath}\"";

ProcessStartInfo psiProbe = new ProcessStartInfo

{

FileName = "ffprobe",

Arguments = argsProbe,

RedirectStandardOutput = true,

UseShellExecute = false,

CreateNoWindow = true

};

using Process probe = new Process { StartInfo = psiProbe };

probe.Start();

string json = await probe.StandardOutput.ReadToEndAsync();

probe.WaitForExit();

using JsonDocument doc = JsonDocument.Parse(json);

var stream = doc.RootElement.GetProperty("streams")[0];

duration = stream.TryGetProperty("duration", out var durEl) ? double.Parse(durEl.GetString()) : 0;

}

string outputFile = Path.Combine(outputDir, $"stream\_{streamIndex}.{extension}");

string args;

if (splitMono)

{

args = $"-y -i \"{filePath}\" -map 0:a:{streamIndex} -filter\_complex channelsplit=channel\_layout=auto -f {extension} \"{outputDir}/%s.{extension}\" -progress pipe:1 -nostats";

}

else

{

args = $"-y -i \"{filePath}\" -map 0:a:{streamIndex} \"{outputFile}\" -progress pipe:1 -nostats";

}

ProcessStartInfo psi = new ProcessStartInfo

{

FileName = "ffmpeg",

Arguments = args,

RedirectStandardOutput = true,

UseShellExecute = false,

CreateNoWindow = true

};

using Process proc = new Process { StartInfo = psi };

proc.Start();

\_ = Task.Run(async () =>

{

using StreamReader reader = proc.StandardOutput;

string line;

while ((line = await reader.ReadLineAsync()) != null)

{

if (line.StartsWith("out\_time\_ms"))

{

string value = line.Split('=')[1];

if (double.TryParse(value, out double msElapsed))

{

double elapsedSeconds = msElapsed / 1\_000\_000.0;

int percent = duration > 0 ? (int)((elapsedSeconds / duration) \* 100) : 0;

int remainingSeconds = (int)(duration - elapsedSeconds);

BeginInvoke(() =>

{

extractionProgressBar.Value = Math.Min(percent, 100);

remainingTimeLabel.Text = $"Remaining: {TimeSpan.FromSeconds(remainingSeconds):mm\\:ss}";

});

}

}

}

});

await proc.WaitForExitAsync();

BeginInvoke(() =>

{

extractionProgressBar.Value = 100;

extractionStatusLabel.Text = "Extraction completed.";

remainingTimeLabel.Text = "";

});

}

private async void Form1\_DragDrop(object sender, DragEventArgs e)

{

if (e.Data.GetDataPresent(DataFormats.FileDrop))

{

string[] files = (string[])e.Data.GetData(DataFormats.FileDrop);

if (files.Length > 0)

{

txtFilePath.Text = files[0];

lstAudioStreams.Items.Clear();

await LoadAudioStreams(files[0]);

}

}

}

private void Form1\_DragEnter(object sender, DragEventArgs e)

{

if (e.Data.GetDataPresent(DataFormats.FileDrop))

e.Effect = DragDropEffects.Copy;

}

}

}

Designer (with drag and drop)

namespace AudioExtractorApp

{

partial class Form1

{

private System.ComponentModel.IContainer components = null;

private System.Windows.Forms.GroupBox groupBox1;

private System.Windows.Forms.Button btnSelectMKV;

private System.Windows.Forms.TextBox txtMKVFilePath;

private System.Windows.Forms.Label lblAudioStreams;

private System.Windows.Forms.ListBox lstAudioStreams;

private System.Windows.Forms.GroupBox groupBox2;

private System.Windows.Forms.RadioButton radioSeparateFiles;

private System.Windows.Forms.RadioButton radioMultichannelFile;

private System.Windows.Forms.GroupBox groupBox3;

private System.Windows.Forms.RadioButton radioWAV;

private System.Windows.Forms.RadioButton radioFLAC;

private System.Windows.Forms.ProgressBar progressBar;

private System.Windows.Forms.Label lblTimeEstimate;

private System.Windows.Forms.Button btnExtractAudio;

private System.Windows.Forms.OpenFileDialog openFileDialog;

protected override void Dispose(bool disposing)

{

if (disposing && (components != null))

components.Dispose();

base.Dispose(disposing);

}

private void InitializeComponent()

{

this.components = new System.ComponentModel.Container();

this.groupBox1 = new System.Windows.Forms.GroupBox();

this.btnSelectMKV = new System.Windows.Forms.Button();

this.txtMKVFilePath = new System.Windows.Forms.TextBox();

this.lblAudioStreams = new System.Windows.Forms.Label();

this.lstAudioStreams = new System.Windows.Forms.ListBox();

this.groupBox2 = new System.Windows.Forms.GroupBox();

this.radioSeparateFiles = new System.Windows.Forms.RadioButton();

this.radioMultichannelFile= new System.Windows.Forms.RadioButton();

this.groupBox3 = new System.Windows.Forms.GroupBox();

this.radioWAV = new System.Windows.Forms.RadioButton();

this.radioFLAC = new System.Windows.Forms.RadioButton();

this.progressBar = new System.Windows.Forms.ProgressBar();

this.lblTimeEstimate = new System.Windows.Forms.Label();

this.btnExtractAudio = new System.Windows.Forms.Button();

this.openFileDialog = new System.Windows.Forms.OpenFileDialog();

//

// groupBox1

//

this.groupBox1.Controls.Add(this.btnSelectMKV);

this.groupBox1.Controls.Add(this.txtMKVFilePath);

this.groupBox1.Location = new System.Drawing.Point(12, 12);

this.groupBox1.Name = "groupBox1";

this.groupBox1.Size = new System.Drawing.Size(400, 70);

this.groupBox1.TabIndex = 0;

this.groupBox1.TabStop = false;

this.groupBox1.Text = "Select MKV File";

//

// btnSelectMKV

//

this.btnSelectMKV.Location = new System.Drawing.Point(320, 28);

this.btnSelectMKV.Name = "btnSelectMKV";

this.btnSelectMKV.Size = new System.Drawing.Size(75, 23);

this.btnSelectMKV.TabIndex = 1;

this.btnSelectMKV.Text = "Browse";

this.btnSelectMKV.UseVisualStyleBackColor = true;

this.btnSelectMKV.Click += new System.EventHandler(this.btnSelectMKV\_Click);

//

// txtMKVFilePath

//

this.txtMKVFilePath.AllowDrop = true;

this.txtMKVFilePath.Location = new System.Drawing.Point(6, 28);

this.txtMKVFilePath.Name = "txtMKVFilePath";

this.txtMKVFilePath.Size = new System.Drawing.Size(300, 20);

this.txtMKVFilePath.TabIndex = 0;

this.txtMKVFilePath.DragEnter += new System.Windows.Forms.DragEventHandler(this.txtMKVFilePath\_DragEnter);

this.txtMKVFilePath.DragDrop += new System.Windows.Forms.DragEventHandler(this.txtMKVFilePath\_DragDrop);

//

// lblAudioStreams

//

this.lblAudioStreams.AutoSize = true;

this.lblAudioStreams.Location = new System.Drawing.Point(12, 85);

this.lblAudioStreams.Name = "lblAudioStreams";

this.lblAudioStreams.Size = new System.Drawing.Size(85, 13);

this.lblAudioStreams.TabIndex = 7;

this.lblAudioStreams.Text = "Audio Streams:";

//

// lstAudioStreams

//

this.lstAudioStreams.FormattingEnabled = true;

this.lstAudioStreams.Location = new System.Drawing.Point(12, 105);

this.lstAudioStreams.Name = "lstAudioStreams";

this.lstAudioStreams.Size = new System.Drawing.Size(400, 95);

this.lstAudioStreams.TabIndex = 6;

this.lstAudioStreams.SelectedIndexChanged += new System.EventHandler(this.lstAudioStreams\_SelectedIndexChanged);

//

// groupBox2

//

this.groupBox2.Controls.Add(this.radioSeparateFiles);

this.groupBox2.Controls.Add(this.radioMultichannelFile);

this.groupBox2.Location = new System.Drawing.Point(12, 206);

this.groupBox2.Name = "groupBox2";

this.groupBox2.Size = new System.Drawing.Size(400, 70);

this.groupBox2.TabIndex = 1;

this.groupBox2.TabStop = false;

this.groupBox2.Text = "Channel Layout";

//

// radioSeparateFiles

//

this.radioSeparateFiles.AutoSize = true;

this.radioSeparateFiles.Location = new System.Drawing.Point(6, 42);

this.radioSeparateFiles.Name = "radioSeparateFiles";

this.radioSeparateFiles.Size = new System.Drawing.Size(138, 17);

this.radioSeparateFiles.TabIndex = 1;

this.radioSeparateFiles.TabStop = true;

this.radioSeparateFiles.Text = "Separate Files (Mono)";

this.radioSeparateFiles.UseVisualStyleBackColor = true;

//

// radioMultichannelFile

//

this.radioMultichannelFile.AutoSize = true;

this.radioMultichannelFile.Location = new System.Drawing.Point(6, 19);

this.radioMultichannelFile.Name = "radioMultichannelFile";

this.radioMultichannelFile.Size = new System.Drawing.Size(135, 17);

this.radioMultichannelFile.TabIndex = 0;

this.radioMultichannelFile.TabStop = true;

this.radioMultichannelFile.Text = "Single Multichannel File";

this.radioMultichannelFile.UseVisualStyleBackColor = true;

//

// groupBox3

//

this.groupBox3.Controls.Add(this.radioWAV);

this.groupBox3.Controls.Add(this.radioFLAC);

this.groupBox3.Location = new System.Drawing.Point(12, 282);

this.groupBox3.Name = "groupBox3";

this.groupBox3.Size = new System.Drawing.Size(400, 70);

this.groupBox3.TabIndex = 2;

this.groupBox3.TabStop = false;

this.groupBox3.Text = "File Format";

//

// radioWAV

//

this.radioWAV.AutoSize = true;

this.radioWAV.Location = new System.Drawing.Point(6, 30);

this.radioWAV.Name = "radioWAV";

this.radioWAV.Size = new System.Drawing.Size(47, 17);

this.radioWAV.TabIndex = 0;

this.radioWAV.TabStop = true;

this.radioWAV.Text = "WAV";

this.radioWAV.UseVisualStyleBackColor = true;

//

// radioFLAC

//

this.radioFLAC.AutoSize = true;

this.radioFLAC.Location = new System.Drawing.Point(6, 53);

this.radioFLAC.Name = "radioFLAC";

this.radioFLAC.Size = new System.Drawing.Size(50, 17);

this.radioFLAC.TabIndex = 1;

this.radioFLAC.TabStop = true;

this.radioFLAC.Text = "FLAC";

this.radioFLAC.UseVisualStyleBackColor = true;

//

// progressBar

//

this.progressBar.Location = new System.Drawing.Point(12, 358);

this.progressBar.Name = "progressBar";

this.progressBar.Size = new System.Drawing.Size(400, 23);

this.progressBar.TabIndex = 3;

//

// lblTimeEstimate

//

this.lblTimeEstimate.AutoSize = true;

this.lblTimeEstimate.Location = new System.Drawing.Point(12, 388);

this.lblTimeEstimate.Name = "lblTimeEstimate";

this.lblTimeEstimate.Size = new System.Drawing.Size(94, 13);

this.lblTimeEstimate.TabIndex = 4;

this.lblTimeEstimate.Text = "Estimated time: N/A";

//

// btnExtractAudio

//

this.btnExtractAudio.Location = new System.Drawing.Point(320, 408);

this.btnExtractAudio.Name = "btnExtractAudio";

this.btnExtractAudio.Size = new System.Drawing.Size(75, 23);

this.btnExtractAudio.TabIndex = 5;

this.btnExtractAudio.Text = "Extract";

this.btnExtractAudio.UseVisualStyleBackColor = true;

this.btnExtractAudio.Click += new System.EventHandler(this.btnExtractAudio\_Click);

//

// openFileDialog

//

this.openFileDialog.Filter = "MKV files (\*.mkv)|\*.mkv";

this.openFileDialog.Title = "Select MKV File";

//

// Form1

//

this.AllowDrop = true;

this.AutoScaleDimensions = new System.Drawing.SizeF(6F, 13F);

this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font;

this.ClientSize = new System.Drawing.Size(424, 441);

this.Controls.Add(this.btnExtractAudio);

this.Controls.Add(this.lblTimeEstimate);

this.Controls.Add(this.progressBar);

this.Controls.Add(this.groupBox3);

this.Controls.Add(this.groupBox2);

this.Controls.Add(this.lstAudioStreams);

this.Controls.Add(this.lblAudioStreams);

this.Controls.Add(this.groupBox1);

this.Name = "Form1";

this.Text = "Audio Extractor";

this.DragEnter += new System.Windows.Forms.DragEventHandler(this.Form1\_DragEnter);

this.DragDrop += new System.Windows.Forms.DragEventHandler(this.Form1\_DragDrop);

this.groupBox1.ResumeLayout(false);

this.groupBox1.PerformLayout();

this.groupBox2.ResumeLayout(false);

this.groupBox2.PerformLayout();

this.groupBox3.ResumeLayout(false);

this.groupBox3.PerformLayout();

this.ResumeLayout(false);

this.PerformLayout();

}

}

}