# time length of an mp3 file

Asked 16 years, 3 months ago  Modified 1 year, 3 months ago

Viewed 32k times

19

What is the simplest way to determine the length (in seconds) of a given mp3 file, **without using outside libraries**? (python source highly appreciated)

python mp3 media duration

Share

Improve this question

Follow

edited May 12, 2021 at 8:14

hippietrail

**16.9k** ● 21 ● 109 ● 173

asked Sep 23, 2008 at 6:33

Silver Dragon

**5,550** ● 7 ● 44 ● 73

5 open the containing folder in Explorer bring up the play time column, screen shot, ORC, text search.... Submit to The Daily WTF /joke – BCS Sep 23, 2008 at 6:35

## 6 Answers

Sorted by: Highest score (default)

You can use pymad. It's an external library, but don't fall for the Not Invented Here trap. Any particular reason you

26

don't want any external libraries?

```
import mad

mf = mad.MadFile("foo.mp3")
track_length_in_milliseconds = mf.total_time()
```

Spotted here.

--

If you really don't want to use an external library, have a look here and check out how he's done it. Warning: it's complicated.

Share Improve this answer

Follow

edited Sep 23, 2008 at 6:56

answered Sep 23, 2008 at 6:37



Harley Holcombe

**182k** ● 15 ● 72 ● 63

I agree with recommending the external library. I haven't used it (or Python for that matter). But I once tried to write, in C++ a program that could simply play MP3s. That didn't pan out, but I did get far enough in to determine the duration of the file. I thought about refactoring that code to ... – Matt Blaine Sep 23, 2008 at 6:50

... post here, but it's pretty darn convoluted. (And in C++, not Python). Not even remotely simple. – Matt Blaine Sep 23, 2008 at 6:52

1 Just a heads up, it looks to me like it only works on certain platforms. The most recent version crashes on installing

because it lacks one of its own setup files, which it recommends I generate by running a second file with linux commands in it. – trevorKirkby Mar 15, 2015 at 0:35

11

For google followers' sake, here are a few more external libs:

- mpg321 -t
- ffmpeg -i
- midentify (mplayer basically) see Using mplayer to determine length of audio/video file
- mencoder (pass it invalid params, it will spit out an error message but also give you info on the file in question, ex $ mencoder inputfile.mp3 -o fake)
- mediainfo program http://mediainfo.sourceforge.net/en
- exiftool
- the linux "file" command
- mp3info
- sox

refs:

- https://superuser.com/questions/36871/linux-command-line-utility-to-determine-mp3-bitrate
- http://www.ruby-forum.com/topic/139468
- mp3 length in milliseconds

(making this a wiki for others to add to).

and libs: .net: naudio, java: jlayer, c: libmad

Cheers!

Share Improve this answer Follow

edited Aug 26, 2023 at 11:15

community wiki
10 revs, 3 users 79%
rogerdpack

of these ffmpeg and mpg321 also handle http links as file locations, unfortunately the latter automatically plays the file, but im perfectly happy with ffmpeg :) – Kami Mar 25, 2011 at 17:27

8

> Simple, parse MP3 binary blob to calculate something, in Python

That sounds like a pretty tall order. I don't know Python, but here's some code I've refactored from another program I once tried to write.

**Note:** It's in C++ (sorry, it's what I've got). Also, as-is, it'll only handle constant bit rate MPEG 1 Audio Layer 3 files. That *should* cover most, but I can't make any guarantee as to this working in all situations. Hopefully this does

what you want, and hopefully refactoring it into Python is easier than doing it from scratch.

```
// determines the duration, in seconds, of an MP3;
// assumes MPEG 1 (not 2 or 2.5) Audio Layer 3 (not 1
// constant bit rate (not variable)

#include <iostream>
#include <fstream>
#include <cstdlib>

using namespace std;

//Bitrates, assuming MPEG 1 Audio Layer 3
const int bitrates[16] = {
          0,  32000,  40000,  48000,  56000,  64000,  8
    112000, 128000, 160000, 192000, 224000, 256000, 32
  };


//Intel processors are little-endian;
//search Google or see: http://en.wikipedia.org/wiki/E
int reverse(int i)
{
    int toReturn = 0;
    toReturn |= ((i & 0x000000FF) << 24);
    toReturn |= ((i & 0x0000FF00) << 8);
    toReturn |= ((i & 0x00FF0000) >> 8);
    toReturn |= ((i & 0xFF000000) >> 24);
    return toReturn;
}

//In short, data in ID3v2 tags are stored as
//"syncsafe integers". This is so the tag info
//isn't mistaken for audio data, and attempted to
//be "played". For more info, have fun Googling it.
int syncsafe(int i)
{
 int toReturn = 0;
 toReturn |= ((i & 0x7F000000) >> 24);
 toReturn |= ((i & 0x007F0000) >>  9);
 toReturn |= ((i & 0x00007F00) <<  6);
```

```
 toReturn |= ((i & 0x0000007F) << 21);
 return toReturn;
}

//How much room does ID3 version 1 tag info
//take up at the end of this file (if any)?
int id3v1size(ifstream& infile)
{
   streampos savePos = infile.tellg();

   //get to 128 bytes from file end
   infile.seekg(0, ios::end);
   streampos length = infile.tellg() - (streampos)128;
   infile.seekg(length);

   int size;
   char buffer[3] = {0};
   infile.read(buffer, 3);
   if( buffer[0] == 'T' && buffer[1] == 'A' && buffer[
     size = 128; //found tag data
   else
     size = 0; //nothing there

   infile.seekg(savePos);

   return size;

}

//how much room does ID3 version 2 tag info
//take up at the beginning of this file (if any)
int id3v2size(ifstream& infile)
{
   streampos savePos = infile.tellg();
   infile.seekg(0, ios::beg);

   char buffer[6] = {0};
   infile.read(buffer, 6);
   if( buffer[0] != 'I' || buffer[1] != 'D' || buffer[
   {
       //no tag data
       infile.seekg(savePos);
       return 0;
   }
```

```
   int size = 0;
   infile.read(reinterpret_cast<char*>(&size), sizeof(
   size = syncsafe(size);

   infile.seekg(savePos);
   //"size" doesn't include the 10 byte ID3v2 header
   return size + 10;
}

int main(int argCount, char* argValues[])
{
  //you'll have to change this
  ifstream infile("C:/Music/Bush - Comedown.mp3", ios:

  if(!infile.is_open())
  {
   infile.close();
   cout << "Error opening file" << endl;
   system("PAUSE");
   return 0;
  }

  //determine beginning and end of primary frame data
  infile.seekg(0, ios::end);
  streampos dataEnd = infile.tellg();

  infile.seekg(0, ios::beg);
  streampos dataBegin = 0;

  dataEnd -= id3v1size(infile);
  dataBegin += id3v2size(infile);

  infile.seekg(dataBegin,ios::beg);

  //determine bitrate based on header for first frame
  int headerBytes = 0;
  infile.read(reinterpret_cast<char*>(&headerBytes),si

  headerBytes = reverse(headerBytes);
  int bitrate = bitrates[(int)((headerBytes >> 12) & 0

  //calculate duration, in seconds
  int duration = (dataEnd - dataBegin)/(bitrate/8);
```

```
  infile.close();

  //print duration in minutes : seconds
  cout << duration/60 << ":" << duration%60 << endl;

  system("PAUSE");
  return 0;
}
```

Share Improve this answer

Follow

edited Oct 5, 2014 at 14:49

wholerabbit

**11.5k** ● 2 ● 38 ● 72

answered Sep 23, 2008 at 7:37

Matt Blaine

**1,976** ● 14 ● 22

1 This works perfectly for something I'm building right now, since I don't need VBR support. All I needed to change was the bitrate since it was assuming 56k when the files were 32k (output from LAME). – alxp Apr 27, 2009 at 9:16

---

**7**

simply use `mutagen`

```
$pip install mutagen
```

use it in python shell:

```
from mutagen.mp3 import MP3
audio = MP3(file_path)
print audio.info.length
```

Share Improve this answer

Follow

edited Mar 22, 2016 at 8:01

answered Mar 13, 2015 at 9:37

Johnny Zhao

**2,898** ● 2 ● 30 ● 26

Best answer. It's easily installed with PIP, no other requirements, and it simply gives the information stored in the file's metadata – Zvika Jul 26, 2016 at 18:55

---

3

Also take a look at audioread (some linux distros including ubuntu have packages), https://github.com/sampsyo/audioread

```
audio = audioread.audio_open('/path/to/mp3')
print audio.channels, audio.samplerate, audio.duration
```

Share Improve this answer

Follow

edited Dec 16, 2013 at 11:01

answered Apr 12, 2012 at 21:26

Bala Clark

**1,524** ● 12 ● 19

---

0

You might count the number of frames in the file. Each frame has a start code, although I can't recollect the exact value of the start code and I don't have MPEG specs

laying around. Each frame has a certain length, around 40ms for MPEG1 layer II.

This method works for CBR-files (Constant Bit Rate), how VBR-files work is a completely different story.

From the document below:

For Layer I files us this formula:

FrameLengthInBytes = (12 * BitRate / SampleRate + Padding) * 4

For Layer II & III files use this formula:

FrameLengthInBytes = 144 * BitRate / SampleRate + Padding

[Information about MPEG Audio Frame Header]

Share Improve this answer

Follow

answered Sep 23, 2008 at 6:43

Mats Wiklander

401 ● 3 ● 10

I believe the length is 26ms. – Ori Pessach Feb 27, 2009 at 21:36