

---

# **pymediainfo Documentation**

***Release 4.2.1***

**Patrick Altman, Louis Sautier**

**Apr 29, 2020**



---

## Contents

---

<b>1</b>	<b>pymediainfo package</b>	<b>1</b>
<b>2</b>	<b>Requirements</b>	<b>5</b>
<b>3</b>	<b>Using MediaInfo</b>	<b>7</b>
<b>4</b>	<b>Reporting Issues / Bugs</b>	<b>9</b>
<b>5</b>	<b>Indices and tables</b>	<b>11</b>
	<b>Python Module Index</b>	<b>13</b>
	<b>Index</b>	<b>15</b>



## 1.1 Module contents

**class** `pymediainfo.MediaInfo(xml, encoding_errors='strict')`

An object containing information about a media file.

*MediaInfo* objects can be created by directly calling code from `libmediainfo` (in this case, the library must be present on the system):

```
>>> pymediainfo.MediaInfo.parse("/path/to/file.mp4")
```

Alternatively, objects may be created from *MediaInfo*'s XML output. Such output can be obtained using the XML output format on versions older than v17.10 and the `OLDXML` format on newer versions.

Using such an XML file, we can create a *MediaInfo* object:

```
>>> with open("output.xml") as f:
...     mi = pymediainfo.MediaInfo(f.read())
```

### Parameters

- **xml** (*str*) – XML output obtained from *MediaInfo*.
- **encoding\_errors** (*str*) – option to pass to `str.encode()`'s *errors* parameter before parsing *xml*.

**Raises** `xml.etree.ElementTree.ParseError` – if passed invalid XML.

**Variables** **tracks** – A list of *Track* objects which the media file contains. For instance:

```
>>> mi = pymediainfo.MediaInfo.parse("/path/to/file.mp4")
>>> for t in mi.tracks:
...     print(t)
<Track track_id='None', track_type='General'>
<Track track_id='1', track_type='Text'>
```

**classmethod** `can_parse` (*library\_file=None*)

Checks whether media files can be analyzed using libmediainfo.

**Return type** `bool`

**classmethod** `parse` (*filename*, *library\_file=None*, *cover\_data=False*, *encoding\_errors='strict'*, *parse\_speed=0.5*, *text=False*, *full=True*, *legacy\_stream\_display=False*, *mediainfo\_options=None*, *output=None*)

Analyze a media file using libmediainfo.

---

**Note:** Because of the way the underlying library works, this method should not be called simultaneously from multiple threads *with different arguments*. Doing so will cause inconsistencies or failures by changing library options that are shared across threads.

---

### Parameters

- **filename** (*str* or *pathlib.Path* or *os.PathLike*) – path to the media file which will be analyzed. A URL can also be used if libmediainfo was compiled with CURL support.
- **library\_file** (*str*) – path to the libmediainfo library, this should only be used if the library cannot be auto-detected.
- **cover\_data** (*bool*) – whether to retrieve cover data as base64.
- **encoding\_errors** (*str*) – option to pass to `str.encode()`'s *errors* parameter before parsing MediaInfo's XML output.
- **parse\_speed** (*float*) – passed to the library as *ParseSpeed*, this option takes values between 0 and 1. A higher value will yield more precise results in some cases but will also increase parsing time.
- **full** (*bool*) – display additional tags, including computer-readable values for sizes and durations.
- **legacy\_stream\_display** (*bool*) – display additional information about streams.
- **mediainfo\_options** (*dict*) – additional options that will be passed to the *MediaInfo\_Option* function, for example: `{"Language": "raw"}`. Do not use this parameter when running the method simultaneously from multiple threads, it will trigger a reset of all options which will cause inconsistencies or failures.
- **output** (*str*) – custom output format for MediaInfo, corresponds to the CLI's `--Output` parameter. Setting this causes the method to return a *str* instead of a *MediaInfo* object.

### Useful values include:

- the empty *str* `" "` (corresponds to the default text output, obtained when running `mediainfo` with no additional parameters)
- `"XML"`
- `"JSON"`
- `%`-delimited templates (see `mediainfo --Info-Parameters`)

**Return type** *str* if *output* is set.

**Return type** *MediaInfo* otherwise.

**Raises**

- **FileNotFoundError** – if passed a non-existent file (Python 3.3), does not work on Windows.
- **IOError** – if passed a non-existent file (Python < 3.3), does not work on Windows.
- **RuntimeError** – if parsing fails, this should not happen unless libmediainfo itself fails.

**Examples:**

```
>>> pymediainfo.MediaInfo.parse("tests/data/sample.mkv")
<pymediainfo.MediaInfo object at 0x7fa83a3db240>
```

```
>>> import json
>>> mi = pymediainfo.MediaInfo.parse("tests/data/sample.mkv",
...     output="JSON")
>>> json.loads(mi)["media"]["track"][0]
{'@type': 'General', 'TextCount': '1', 'FileExtension': 'mkv',
 'FileSize': '5904', ... }
```

**to\_data()**

Returns a dict representation of the object's *Tracks*.

**Return type** dict

**to\_json()**

Returns a JSON representation of the object's *Tracks*.

**Return type** str

**class** pymediainfo.**Track**(*xml\_dom\_fragment*)

An object associated with a media file track.

Each *Track* attribute corresponds to attributes parsed from MediaInfo's output. All attributes are lower case. Attributes that are present several times such as Duration yield a second attribute starting with *other\_* which is a list of all alternative attribute values.

When a non-existing attribute is accessed, *None* is returned.

Example:

```
>>> t = mi.tracks[0]
>>> t
<Track track_id='None', track_type='General'>
>>> t.duration
3000
>>> t.to_data()["other_duration"]
['3 s 0 ms', '3 s 0 ms', '3 s 0 ms',
 '00:00:03.000', '00:00:03.000']
>>> type(t.non_existing)
NoneType
```

All available attributes can be obtained by calling *to\_data()*.

**to\_data()**

Returns a dict representation of the track attributes.

Example:

```
>>> sorted(track.to_data().keys())[:3]
['codec', 'codec_extensions_usually_used', 'codec_url']
>>> t.to_data()["file_size"]
5988
```

**Return type** dict

## CHAPTER 2

---

### Requirements

---

This is a simple wrapper around the MediaInfo library, which you can find at <https://mediaarea.net/en/MediaInfo>

Binary wheels containing the library are provided for Windows and Mac OS X.

Packages are available for [several Linux distributions](#).



## CHAPTER 3

---

### Using MediaInfo

---

There isn't much to this library so instead of a lot of documentation it is probably best to just demonstrate how it works:

```
from pymediainfo import MediaInfo
media_info = MediaInfo.parse('my_video_file.mov')
for track in media_info.tracks:
    if track.track_type == 'Video':
        print(track.bit_rate, track.bit_rate_mode, track.codec)

# output: 46033920 CBR DV
```

If you already have the XML data in a string in memory (e.g. you have previously parsed the file or were sent the dump from *mediainfo* from someone else) you can call the constructor directly:

```
from pymediainfo import MediaInfo
media_info = MediaInfo(raw_xml_string)
```

Since the attributes on the *Track* objects are being dynamically added as the XML output from *MediaInfo* is being parsed, there isn't a firm definition of what will be available at runtime. In order to make consuming the objects easier so that you can avoid having to use *hasattr* or *try/except* blocks, the `__getattr__` method has been overridden and will just return *None* when and if an attribute is referenced but doesn't exist.

This will enable you to write consuming code like:

```
from pymediainfo import MediaInfo
media_info = MediaInfo.parse('my_video_file.mov')
for track in media_info.tracks:
    if track.bit_rate is not None:
        print("{}: {}".format(track.track_type, track.bit_rate))
    else:
        print("{} tracks do not have bit rate associated with them.".format(track.track_type))
```

Output:

```
General tracks do not have bit rate associated with them.  
Video: 46033920  
Audio: 1536000  
Menu tracks do not have bit rate associated with them.
```

## CHAPTER 4

---

### Reporting Issues / Bugs

---

Please use the issue tracker in GitHub at <https://github.com/sbraz/pymediainfo/issues> to report all feature requests or bug reports. Thanks!



## CHAPTER 5

---

### Indices and tables

---

- `genindex`
- `modindex`
- `search`



**p**

`pymediainfo`, [1](#)



### C

`can_parse()` (*pymediainfo.MediaInfo class method*), [1](#)

### M

`MediaInfo` (*class in pymediainfo*), [1](#)

### P

`parse()` (*pymediainfo.MediaInfo class method*), [2](#)  
`pymediainfo` (*module*), [1](#)

### T

`to_data()` (*pymediainfo.MediaInfo method*), [3](#)  
`to_data()` (*pymediainfo.Track method*), [3](#)  
`to_json()` (*pymediainfo.MediaInfo method*), [3](#)  
`Track` (*class in pymediainfo*), [3](#)