Python AVI Parser

Project Code Documentation

Matteo Nesti
m.nesti@gmail.com

Aurel Pjetri
pjetri.aurel@gmail.com

May 9, 2018

Abstract

The goal of this project is to implement an AVI parser able to explore the file's tree-like structure and, as a second step, write an XML representation of the retrieved information in a specific output file.

1 Introduction

In this report we present the parser's inner mode of operation and give a description of the main methods' functionality.

This project is composed of 4 files:

- `avi_chunk.py` contains the classes used for data representation.
- `chunk_fields.py` contains known chunks' structure.
- `main.py` contains the principal methods for starting the parser and generating the XML file.
- `utils.py` contains support methods for fields' parsing.

The execution of this project is organized into two consecutive phases. The first step is to parse the AVI file and create the objects structure. The second step instead consists of exploring the constructed structure and generating the XML tree.

For the implementation of this project we used Python 3.6.2 (it should work also with other versions of Python, but was not tested). In particular, in order to complete the tasks of parsing the AVI file and writing the

XML, we used modules from the Python standard library such as `chunk` and `xml.etree.cElementTree`.

For completeness, in table 1 we report all used modules and their versions.

`Python`	3.6.2
`chunk`	-
`xml.etree.cElementTree`	1.3.0
`unicodedata`	9.0.0
`argparse`	1.1

Table 1: Used modules

2 Data Representation

For the parsed data representation two classes were used: `LeafChunk` and `ListChunk`. Those classes present all attributes and methods for storing and retrieving the needed information.

2.1 LeafChunk

`LeafChunk` is the class used to contain informations parsed from a "basic" chunk (non-LIST or RIFF chunk). It has the basic name and size attributes. Moreover it presents a dictionary attribute to store the known fields represented in the `chunk_fields.py` file and a second attribute for unkown raw data.

2.2 ListChunk

`ListChunk` is used to contain informations related to LIST and RIFF chunks. It has a dedicated attribute for size, name and type. It keeps also track of its subchunks by storing them in a dedicated list.

3 Parsing

For parsing the AVI file three files are involved: `main.py`, `utils.py` and `chunk_fields.py`.

In the `main.py` file we can find two of the core methods for parsing: `search_list` and `main`. Moreover `utils.py` contains all the other required

methods and finally `chunk_fields.py` contains the reference dictionary for fields parsing as decribed in Section 3.3.

In the main method we open the avi file, instantiate the `ListChunk` object for the `RIFF 'AVI '` chunk and start the recurrent call of the `search_list` function.

3.1 search_list

This function is used to search within the file's tree structure.

If the found chunk is itself a LIST chunk a ListChunk object is instantiated and a recursive call to this function is made. Finally the returned chunks from the call are added to the object.

On the other hand, if the bytes that are going to be explored by the function `utils.parse_chunk_data` exceed the limit (the size of the current chunk's parent), the call is not made and the file pointer moves back (`file.seek(-8,1)`) as if the 8 bytes (2 fields) were not ever read. Otherwise, a LeafChunk is instantiated, `parse_chunk_data` function from `utils.py` is called and the collected fields are added to the object.

The core part of the function is presented below.

```
while (bytes_explored<limit):
    ...
        # if it's a LIST chunk call recurivelly this function
        if current_chunk_name == b'LIST':
            current_chunk_type = current_chunk.read(4)
            explored_chunk = ListChunk(current_chunk_name, current_chunk_size, current_chunk_type)
            subchunks = search_list(file, current_chunk_size - 4)
            explored_chunk.add_subchunks(subchunks)

        # if it's a basic chunk call the utils function to collect its fields ('parse_chunk_data')
        elif bytes_explored + current_chunk_size + 8 <= limit:
            explored_chunk = LeafChunk(current_chunk_name, current_chunk_size)

            utils.parse_chunk_data(file, current_chunk_name.decode('utf-8'), current_chunk, explored_chunk, current_chunk_size)

        # to avoid the chunk to be inserted in the wrong LIST chunk
        else:
            file.seek(-8,1)
```

```
    ...
```

3.2 utils.parse_chunk_data

In this method we check if the given chunk has a known structure by checking the dictionary in `chunk_fields.py`. For unkown chunks, raw data in bytes format is collected and returned. Otherwise three cases are managed: index and stream format chunks that are handled respectivelly with `utils.parse_index` and `utils.parse_strf` and other known chunks handled with `utils.collect_fields`.

We present the core part of the function

```
...
# index chunks
if chunk_name == 'idx1' or chunk_name == 'indx' or chunk_name
    [:2] == 'ix' or chunk_name[-2:] == 'ix':
    parse_index(file, chunk_name, current_chunk, lc, limit,
    _acfd)

# stream format chunks
elif chunk_name == 'strf':
    parse_strf(file, _acfd, current_chunk, lc, limit)

# unknown chunk
elif chunk_name not in [k[:4] for k in list(_acfd.keys())]:
    raw_data = current_chunk.read(limit)
    lc.add_rawdata(raw_data)

# other known chunks
else:
    collected_fields, r = collect_fields(file,0, limit,
    current_chunk, _acfd[chunk_name])
...
```

3.3 chunk_fields.py

All the known chunks and their fields are collected in a specific dictionary (`_acfd`) defined in the `chunk_fields.py` file. It is used by the parsing functions to correctly interpret the bytes read from the file.

In order to achieve the maximum generality and maintainability possible we defined a specific syntax for the chunk structure's definition. This way it's possible to extend the parser's known chunks by adding its definition as a new entry in the `_acfd` dictionary.

In this dictionary every known chunk's name is key to an other dictionary with its fields. Moreover in this inner dictionary every field is key to the number of bytes occupied by it. So we have:

```
'chunk id': {
    'field name': bytes,
    'field name': bytes
    ...}
```

There are three special cases: struct fields, fcc fields, unknown fields.

Struct Fields

Some fields can be structs, in this case its value is an annidated dictionary with the same sintax. For example:

```
{...,
    'dwReserved':{
        'dwReserved_1':4,
        'dwReserved_2':4,
        'dwReserved_3':4,
        'dwReserved_4':4}}
```

This fields appear in `avih` chunk:

```
'avih':{
        'dwMicroSecPerFrame': 4,
        'dwMaxBytesPerSec': 4,
        'dwPaddingGranularity': 4,
        'dwFlags': 4,
        'dwTotalFrames': 4,
        'dwInitialFrames': 4,
        'dwStreams': 4,
        'dwSuggestedBufferSizeWidth': 4,
        'dwWidth': 4,
        'dwHeight': 4,
        'dwReserved':{
            'dwReserved_1':4,
            'dwReserved_2':4,
            'dwReserved_3':4,
            'dwReserved_4':4}}
```

A notable subcase are **arrays with undefined number of elements**. There may be, infact, arrays whose entries are structs. Moreover the number of array members is not explicitly declared. In this case we use two annidated

dictionaries : the outer dictionary has only one key that is the identifier of the struct entry and the inner specifies the struct's structure.

```
{...,
    'aIndex':{
            '_avisuperindex_entry': {
                'qwOffset':8,
                'dwSize':4,
                'dwDuration':4}}}
```

Where the complete chunk is:

```
'indx':{
        'wLongsPerEntry':2,
        'bIndexSubType':1,
        'bIndexType':1,
        'nEntriesInUse':4,
        'dwChunkId':'fcc',
        'dwReserved':{
            'dwReserved_1':4,
            'dwReserved_2':4,
            'dwReserved_3':4},
        'aIndex':{
            '_avisuperindex_entry': {
                'qwOffset':8,
                'dwSize':4,
                'dwDuration':4}}}
```

Fcc Fields

There are some fields whose value is a FourCC, therefore a 4 byte string. In these cases we replace the bytes value with the 'fcc' keyword. This way the parser will read 4 bytes and decode them with the UTF-8 codec.

```
'strh':{
        'fccType': 'fcc',
        'fccHandler': 'fcc',
        'dwFlags': 4,
        ...}
```

Unknown Fields

There can be some fields whose content and size is unkown. In this case we use the keyword 'unknown' and the value '-1'. This case may occur only in

the final part of the chunk therefore with the -1 value the parser will read all the remaining bytes and insert them in the 'unkown' field.

4 XML Generation

The XML file is generated exploring the built object structure during the parsing phase. In the `main` method the node associated to the RIFF chunk is generated and the `populated_xml` method is called.

In `populate_xml` whenerver an atom (LeafChunk or ListChunk) is found an XML node is instantiated. For ListChunks a recursive call to the `populate_xml` function is made. In case of a LeafChunk the collected fields are added as attributes of the XML node. Before being written in the XML all field values are cleaned up from utf-8 control characters.

```
def remove_control_chars(s):
    r = ''
    warning = False
    for ch in s:
        if category(ch)[0] != "C":
            r += ch
        else:
            warning = True
    return r, warning
```

A particular case is managed in the movi list where some data chunks can be found having a different identifier. While the base syntax expects the first two bytes to be digits (representing the stream to which the data chunk is referred) and the second two to be characters (representing the type of the stream), the special case (introduced by the OpenDML extension) presents those pairs of bytes in swapped places.

The following is the Python code that handles this kind of cases:

```
# subindex special case: some files can have indexes
# identified for example as 'ix01' instead of '01ix'
if name[0] and name[1] in string.ascii_letters:
    if verbose:
        log_s = "Managed chunk #{} : that is {} instead of
    standard style: ##CC ".format(str(child), name)
        print(log_s)
        log_strings.append(log_s)
    nameChars = name[:2]
    nameDigits = name[2:]
if show_data_chunks or ( (not show_data_chunks) and (
    nameChars == 'ix') ):
    node = ET.SubElement(et_parent, nameChars + '-' + str(
    child)) node.set("stream", nameDigits)
```

Finally in the main method a check is made to test if the data contained in the RIFF chunk has the same size of the file. If not, an additional XML node is added containing the raw data in excess.

All performed changes in this phase can be notified with a warning message and registered in a specifically generated log file if the related `verbose` flag is enables.

Finally there is the possibility to avoid the writing of all the stream data subchunks in the movi list for a more readable XML setting the specific `show_data_chunks` flag to false.

4.1 XML Output Examples

We report some examples of possible outputs.

Base Case

This is an example of an AVI file in which no warning was thrown. Therefore no log file was generated and the only output is the presented XML. This means that in this case all 'movi' LIST subchunks had a regular syntax and parsed fields didn't contain control characters.

```
<root filename="sample.avi" size="375688">
    <RIFF size="375680" type="AVI ">
        <LIST-1 size-1="382" type-0="hdrl">
            <avih-1 dwFlags-4="2320" dwHeight-10="240"
   dwInitialFrames-6="0" dwMaxBytesPerSec-2="0"
   dwMicroSecPerFrame-1="40000" dwPaddingGranularity-3="0"
   dwReserved-11="{'dwReserved_1': 0, 'dwReserved_2': 0, '
   dwReserved_3': 0, 'dwReserved_4': 0}" dwStreams-7="2"
   dwSuggestedBufferSizeWidth-8="0" dwTotalFrames-5="139"
   dwWidth-9="320" size-0="56"/>
            <LIST-2 size-1="192" type-0="strl">
                <strh-1 dwFlags-3="0" dwInitialFrames-6="0"
   dwLength-10="139" dwQuality-12="0" dwRate-8="25"
   dwSampleSize-13="0" dwScale-7="1" dwStart-9="0"
   dwSuggestedBufferSize-11="6583" fccHandler-2="XVID"
   fccType-1="vids" rcFrame-14="{'left': 0, 'top': 0, 'right
   ': 320, 'bottom': 240}" size-0="56" wLanguage-5="0"
   wPriority-4="0"/>
                <strf-2 biBitCount-5="12" biClrImportant-11="
   0" biClrUsed-10="0" biCompression-6="XVID" biHeight-3="240
   " biPlanes-4="1" biSize-1="40" biSizeImage-7="115200"
   biWidth-2="320" biXPelsPerMeter-8="0" biYPelsPerMeter-9="0
   " bmiColors-12="{}" size-0="40" unknown-13="b''"/>
                <vprp-3 FieldInfo-10="{'nbFieldPerFrame-0':
   {'CompressedBMHeight': 240, 'CompressedBMWidth': 320, '
```

```
ValidBMHeight': 240, 'ValidBMWidth': 320, 'ValidBMXOffset
': 0, 'ValidBMYOffset': 0, 'VideoXOffsetInT': 0, '
VideoYValidStartLine': 0}}" VideoFormatToken-1="0"
VideoStandard-2="0" dwFrameAspectRatio-6="262147"
dwFrameHeightInPixels-8="240" dwFrameWidthInPixels-7="320"
 dwHTotalInT-4="320" dwVTotalInLines-5="240"
dwVerticalRefreshRate-3="25" nbFieldPerFrame-9="1" size-0=
"68"/>
            </LIST-2>
            <LIST-3 size-1="106" type-0="strl">
                <strh-1 dwFlags-3="0" dwInitialFrames-6="20"
dwLength-10="211" dwQuality-12="0" dwRate-8="44100"
dwSampleSize-13="0" dwScale-7="1152" dwStart-9="0"
dwSuggestedBufferSize-11="522" fccHandler-2="U" fccType-1=
"auds" rcFrame-14="{'left': 0, 'top': 0, 'right': 0, '
bottom': 0}" size-0="56" wLanguage-5="0" wPriority-4="0"/>
                <strf-2 cbSize-7="12" fdwFlags-9="2"
nAvgBytesPerSec-4="12548" nBlockAlign-5="1152" nBlockSize
-10="1152" nChannels-2="2" nCodecDelay-12="0"
nFramesPerBlock-11="1" nSamplesPerSec-3="44100" size-0="30
" wBitsPerSample-6="0" wFormatTag-1="85" wID-8="1"/>
            </LIST-3>
        </LIST-1>
        <LIST-2 size-1="38" type-0="INFO">
            <ISFT-1 Software-1="b'MEncoder SVN-r33148-4.0.1\
x00'" size-0="26"/>
        </LIST-2>
        <JUNK-3 rawData-1="b'[= MPlayer junk data! =][=
MPlayer junk data! =] ... [= MPlayer junk data! =][=
MPlayer junk '" size-0="3640"/>
        <LIST-4 size-1="365976" type-0="movi">
            <wb-1 stream="01"/>
            <wb-2 stream="01"/>
            <wb-3 stream="01"/>
            <wb-4 stream="01"/>
            ...
            <wb-20 stream="01"/>
            <dc-21 stream="00"/>
            <wb-22 stream="01"/>
            <dc-23 stream="00"/>
            <wb-24 stream="01"/>
            <wb-25 stream="01"/>
            <dc-26 stream="00"/>
            <wb-27 stream="01"/>
            <dc-28 stream="00"/>
            <wb-29 stream="01"/>
            <wb-30 stream="01"/>
            <dc-31 stream="00"/>
            ...
```

```
38              <dc-349 stream="00"/>
39              <dc-350 stream="00"/>
40          </LIST-4>
41          <idx1-5 aIndex-1="{'_avioldindex_entry-0': {'
    dwChunkId': '01wb', 'dwFlags': 16, 'dwOffset': 4, 'dwSize
    ': 313}, '_avioldindex_entry-1': {'dwChunkId': '01wb', '
    dwFlags': 16, 'dwOffset': 326, 'dwSize': 313}, ... , '
    _avioldindex_entry-348': {'dwChunkId': '00dc', 'dwFlags':
    0, 'dwOffset': 357940, 'dwSize': 4336}, '
    _avioldindex_entry-349': {'dwChunkId': '00dc', 'dwFlags':
    0, 'dwOffset': 362284, 'dwSize': 3683}}" size-0="5600"/>
42      </RIFF>
43  </root>
```

Movi List Subchunks Warning

The second is an example where the special case in the movi list is present. We therefore present the output XML and the generated log file with the warning messages.

Reading the log file we can see that there where two subindex chunks with 'ix00' and 'ix01' identifiers instead of the standard '00ix' and '01ix'. As we can see this deviation from the base case is handled in the XML and reported in the log file.

```
1  <root filename="MVI_1100.avi" size="940804">
2      <RIFF size="940796" type="AVI ">
3          <LIST-1 size-1="850" type-0="hdrl">
4              <avih-1 dwFlags-4="65552" dwHeight-10="480"
    dwInitialFrames-6="0" dwMaxBytesPerSec-2="1067564"
    dwMicroSecPerFrame-1="33333" dwPaddingGranularity-3="0"
    dwReserved-11="{'dwReserved_1': 0, 'dwReserved_2': 0, '
    dwReserved_3': 0, 'dwReserved_4': 0}" dwStreams-7="2"
    dwSuggestedBufferSizeWidth-8="35218" dwTotalFrames-5="30"
    dwWidth-9="640" size-0="56"/>
5              <LIST-2 size-1="244" type-0="strl">
6                  <strh-1 dwFlags-3="0" dwInitialFrames-6="0"
    dwLength-10="30" dwQuality-12="10000" dwRate-8="1000000"
    dwSampleSize-13="0" dwScale-7="33333" dwStart-9="0"
    dwSuggestedBufferSize-11="35218" fccHandler-2="mjpg"
    fccType-1="vids" rcFrame-14="{'left': 0, 'top': 0, 'right
    ': 640, 'bottom': 480}" size-0="56" wLanguage-5="0"
    wPriority-4="0"/>
7                  <strf-2 biBitCount-5="24" biClrImportant-11="
    0" biClrUsed-10="0" biCompression-6="MJPG" biHeight-3="480
    " biPlanes-4="1" biSize-1="40" biSizeImage-7="921600"
    biWidth-2="640" biXPelsPerMeter-8="0" biYPelsPerMeter-9="0
    " bmiColors-12="{}" size-0="40" unknown-13="b''"/>
```

```
8               <indx-3 aIndex-7="{'_avisuperindex_entry-0':
  {'qwOffset': 940452, 'dwSize': 272, 'dwDuration': 30}, '
  _avisuperindex_entry-1': {'qwOffset': 0, 'dwSize': 0, '
  dwDuration': 0}, '_avisuperindex_entry-2': {'qwOffset': 0,
   'dwSize': 0, 'dwDuration': 0}, '_avisuperindex_entry-3':
  {'qwOffset': 0, 'dwSize': 0, 'dwDuration': 0}, '
  _avisuperindex_entry-4': {'qwOffset': 0, 'dwSize': 0, '
  dwDuration': 0}, '_avisuperindex_entry-5': {'qwOffset': 0,
   'dwSize': 0, 'dwDuration': 0}}" bIndexSubType-2="0"
  bIndexType-3="0" dwChunkId-5="00dc" dwReserved-6="{'
  dwReserved_1': 0, 'dwReserved_2': 0, 'dwReserved_3': 0}"
  nEntriesInUse-4="1" size-0="120" wLongsPerEntry-1="4"/>
9           </LIST-2>
10          <LIST-3 size-1="220" type-0="strl">
11              <strh-1 dwFlags-3="0" dwInitialFrames-6="0"
  dwLength-10="11024" dwQuality-12="10000" dwRate-8="11024"
  dwSampleSize-13="1" dwScale-7="1" dwStart-9="0"
  dwSuggestedBufferSize-11="11024" fccHandler-2="" fccType-1
  ="auds" rcFrame-14="{'left': 0, 'top': 0, 'right': 0, '
  bottom': 0}" size-0="56" wLanguage-5="0" wPriority-4="0"/>
12              <strf-2 cbSize-7="0" nAvgBytesPerSec-4="11024
  " nBlockAlign-5="1" nChannels-2="1" nSamplesPerSec-3="
  11024" size-0="16" unknown-8="b''" wBitsPerSample-6="8"
  wFormatTag-1="1"/>
13              <indx-3 aIndex-7="{'_avisuperindex_entry-0':
  {'qwOffset': 940724, 'dwSize': 40, 'dwDuration': 11024}, '
  _avisuperindex_entry-1': {'qwOffset': 0, 'dwSize': 0, '
  dwDuration': 0}, '_avisuperindex_entry-2': {'qwOffset': 0,
   'dwSize': 0, 'dwDuration': 0}, '_avisuperindex_entry-3':
  {'qwOffset': 0, 'dwSize': 0, 'dwDuration': 0}, '
  _avisuperindex_entry-4': {'qwOffset': 0, 'dwSize': 0, '
  dwDuration': 0}, '_avisuperindex_entry-5': {'qwOffset': 0,
   'dwSize': 0, 'dwDuration': 0}}" bIndexSubType-2="0"
  bIndexType-3="0" dwChunkId-5="01wb" dwReserved-6="{'
  dwReserved_1': 0, 'dwReserved_2': 0, 'dwReserved_3': 0}"
  nEntriesInUse-4="1" size-0="120" wLongsPerEntry-1="4"/>
14          </LIST-3>
15          <LIST-4 size-1="260" type-0="odml">
16              <dmlh-1 dwTotalFrames-1="30" size-0="248"
  unknown-2="b'\x00\x00...\x00\x00'"/>
17          </LIST-4>
18          <IDIT-5 rawData-1="b'FRI AUG 03 18:30:04 2012\n\
  x00'" size-0="26"/>
19      </LIST-1>
20      <LIST-2 size-1="24" type-0="INFO">
21          <ISFT-1 Software-1="b'CanonMVI06\x00\x00'" size-0
  ="12"/>
22      </LIST-2>
23      <JUNK-3 rawData-1="b'\x00\x00...\x00\x00'" size-0="
```

```
    1138"/>
24          <LIST-4 size-1="938708" type-0="movi">
25              <wb-1 stream="01"/>
26              <dc-2 stream="00"/>
27              <dc-3 stream="00"/>
28              <dc-4 stream="00"/>
29              ...
30              <dc-31 stream="00"/>
31              <ix-32 stream="00"/>
32              <ix-33 stream="01"/>
33          </LIST-4>
34          <idx1-5 aIndex-1="{'_avioldindex_entry-0': {'
    dwChunkId': '01wb', 'dwFlags': 0, 'dwOffset': 4, 'dwSize':
     11024}, '_avioldindex_entry-1': {'dwChunkId': '00dc', '
    dwFlags': 16, 'dwOffset': 11036, 'dwSize': 14190}}" size-0
    ="32"/>
35      </RIFF>
36  </root>
```

Generated log file:

```
1  Managed chunk #32 : that is ix00 instead of standard style:
       ##CC
2  Managed chunk #33 : that is ix01 instead of standard style:
       ##CC
```

Control Characters Warning

This third example shows the case of a file with fields containing control characters. As in the previous example we present the two outputted files (XML and log).

```
1  <root filename="metaxas.avi" size="36484160">
2      <RIFF size="36484152" type="AVI ">
3          <LIST-1 size-1="8830" type-0="hdrl">
4              <avih-1 dwFlags-4="2320" dwHeight-10="99"
    dwInitialFrames-6="0" dwMaxBytesPerSec-2="25000"
    dwMicroSecPerFrame-1="66666" dwPaddingGranularity-3="0"
    dwReserved-11="{'dwReserved_1': 0, 'dwReserved_2': 0, '
    dwReserved_3': 0, 'dwReserved_4': 0}" dwStreams-7="2"
    dwSuggestedBufferSizeWidth-8="1048576" dwTotalFrames-5="
    20282" dwWidth-9="176" size-0="56"/>
5              <LIST-2 size-1="4244" type-0="strl">
6                  <strh-1 dwFlags-3="0" dwInitialFrames-6="0"
    dwLength-10="20282" dwQuality-12="4294967295" dwRate-8="15
    " dwSampleSize-13="0" dwScale-7="1" dwStart-9="0"
    dwSuggestedBufferSize-11="8689" fccHandler-2="FMP4"
    fccType-1="vids" rcFrame-14="{'left': 0, 'top': 0, 'right
```

```
': 176, 'bottom': 99}" size-0="56" wLanguage-5="0"
wPriority-4="0"/>
                <strf-2 biBitCount-5="24" biClrImportant-11="
0" biClrUsed-10="0" biCompression-6="FMP4" biHeight-3="99"
 biPlanes-4="1" biSize-1="40" biSizeImage-7="52272"
biWidth-2="176" biXPelsPerMeter-8="0" biYPelsPerMeter-9="0
" bmiColors-12="{}" size-0="40" unknown-13="b''"/>
                <JUNK-3 rawData-1="b'\x04\x00...\x00'" size-0
="4120"/>
            </LIST-2>
            <LIST-3 size-1="4234" type-0="strl">
                <strh-1 dwFlags-3="0" dwInitialFrames-6="0"
dwLength-10="37563" dwQuality-12="4294967295" dwRate-8="
250" dwSampleSize-13="0" dwScale-7="9" dwStart-9="0"
dwSuggestedBufferSize-11="108" fccHandler-2="" fccType-1="
auds" rcFrame-14="{'left': 0, 'top': 0, 'right': 0, '
bottom': 0}" size-0="56" wLanguage-5="0" wPriority-4="0"/>
                <strf-2 cbSize-7="12" fdwFlags-9="2"
nAvgBytesPerSec-4="0" nBlockAlign-5="576" nBlockSize-10="
1152" nChannels-2="1" nCodecDelay-12="1393"
nFramesPerBlock-11="1" nSamplesPerSec-3="16000" size-0="30
" wBitsPerSample-6="0" wFormatTag-1="85" wID-8="1"/>
                <JUNK-3 rawData-1="b'\x04\x00\x00...\x00\x00'
" size-0="4120"/>
            </LIST-3>
            <JUNK-4 rawData-1="b'odmldmlh\xf8\x00\x00\x00...\
x00'" size-0="260"/>
        </LIST-1>
        <LIST-2 size-1="26" type-0="INFO">
            <ISFT-1 Software-1="b'Lavf57.66.105\x00'" size-0=
"14"/>
        </LIST-2>
        <JUNK-3 rawData-1="b'\x00...\x00\x00'" size-0="1016"/
>
        <LIST-4 size-1="35548716" type-0="movi">
            <wb-1 stream="01"/>
            <wb-2 stream="01"/>
            <dc-3 stream="00"/>
            <wb-4 stream="01"/>
            ...
            <wb-57844 stream="01"/>
            <wb-57845 stream="01"/>
        </LIST-4>
        <idx1-5 aIndex-1="{'_avioldindex_entry-0': {'
dwChunkId': '01wb', 'dwFlags': 16, 'dwOffset': 4, 'dwSize
': 108}, '_avioldindex_entry-1': {'dwChunkId': '01wb', '
dwFlags': 16, 'dwOffset': 120, 'dwSize': 108}, ..., '
_avioldindex_entry-57844': {'dwChunkId': '01wb', 'dwFlags
': 16, 'dwOffset': 35548600, 'dwSize': 108}}" size-0="
```

```
    925520"/>
31   </RIFF>
32 </root>
```

Generated log file:

```
1 Removed control characters in chunk 'strh' (#1): field '
    fccHandler' (#1) value was (in bytes): b'\x01\x00\x00\x00'
     wrote: b'' (decoded) instead
```

5 Running The Project

By default the project has both its verbosity and movie data chunks flags set to true. Moreover given a *example.avi* file the output XML and log files will be named by default *example-tree.xml* and *example-log.txt* respectively and will be stored in two specific folders named 'xmls' and 'logs'.

To run the project with these default settings the first step is to create two folders in the the directory containing the `main.py` file named 'xmls' and 'logs' that will contain respectively the outputted XML and log files.

Then from a terminal, after having moved to the `main.py` file directory execute the following command:

```
1 python main.py file_path.avi
```

In order to see all the possible settings print the help dialog:

```
1 python main.py -h
```

And this will be the output:

```
1 usage: main.py [-h] [-v VERBOSE] [-o OUT] [-l LOG] [-m MOVI]
    [-xd XMLDIR]
2                [-ld LOGDIR]
3                path
4
5 positional arguments:
6   path                  path to the AVI file
7
8 optional arguments:
9   -h, --help            show this help message and exit
10  -v VERBOSE, --verbose VERBOSE
11                        Verbosity flag for warnings (default:
      True)
12  -o OUT, --out OUT     XML filename (default: <AVI filename
    >-tree.xml)
13  -l LOG, --log LOG     log filename (default: <AVI filename
    >-log.txt)
14  -m MOVI, --movi MOVI  flag to show stream chunks in movi
    list (default:
```

```
                        True)
  -xd XMLDIR, --xmldir XMLDIR
                        directory path for the generated xml
   (default:
                        ./xmls/)
  -ld LOGDIR, --logdir LOGDIR
                        directory path for the log eventually
    generated
                        (default: ./logs/)
```

As we can see other than the possibility to disable the two flags, the user can also specify the XML and log file names and the folders in which to save them.