AVI File Format

Last change: December 14, 2006

Contents

1 Introduction

1.1 Why another AVI file format documentation?

Even though the AVI file format has been around for more than 10 years, there is no documentation available which does not only describe the format itself, but which also informs about issues that come from flawed demuxers and flawed decoders, and how to circumvent them.

The goal of this document is not only to explain the AVI format, as it is defined on the paper, but rather how to use it, when working with flawed muxers, flawed demuxers, and flawed decompressors.

1.2 Basic data structures

There are 2 types of atoms in AVI files:

1.2.1 Chunks

```
typedef struct {
  DWORD  dwFourCC
  DWORD  dwSize
  BYTE   data[dwSize]    // contains headers or video/audio data
} CHUNK;
```

1.2.2 Lists

```
typedef struct {
  DWORD  dwList
  DWORD  dwSize
  DWORD  dwFourCC
  BYTE   data[dwSize-4]  // contains Lists and Chunks
} LIST;
```

A chunk containing video, audio or subtitle data uses a `dwFourCC` containing 2 hexadecimal digits specifying the stream number and 2 letters specifying the data type (dc = video, wb = audio, tx = text). The values `dwFourCC` and `dwSize` have the same meaning in both of the structures:
`dwFourCC` describes the type of the chunk (for example `'hdrl'` for 'header list'), and `dwSize` contains the size of the chunk or list, including the first byte after the `dwSize` value. In the case of Lists, this includes the 4 bytes taken by `dwFourCC`!

The value of `dwList` can be `'RIFF'` ('RIFF-List') or `'LIST'` ('List').

1.3 AVI file types

Basicly, there are 3 types of AVI files:

- **AVI 1.0** The original, old AVI file.
- **Open-DML** An extension to the AVI file format. Version 1.02 has been specified by 28/02/1996. The most important improvements are:
  - almost unlimited filesize (much more than what NTFS allows, for example)
  - overhead reduced by 33%
- **Hybride-Files**: Open-DML files that contain an additional Legacy Index for compatibility reasons. This is not an "official" word for those files, but it is describing that file type pretty well. Hybride files containing only one RIFF List can be treated as either file type.

This document describes the subset of Open-DML 1.02 file format features, as well as some additions ('hacks'), which work in common players if the proper (freely available) filters are installed. Features that work in Open-DML, but not in AVI 1.0, will be indicated to be Open-DML only.

2 Layout of an AVI file

A `RIFF-List` where `dwFourCC = 'AVI '` shall be called a 'RIFF-AVI-List', a `RIFF-List` where `dwFourCC = 'AVIX'` shall be called a 'RIFF-AVIX-List'.

Every AVI file has the following layout:

```
RIFF AVI        // mandatory
{ RIFF AVIX }   // only for Open-DML files
```

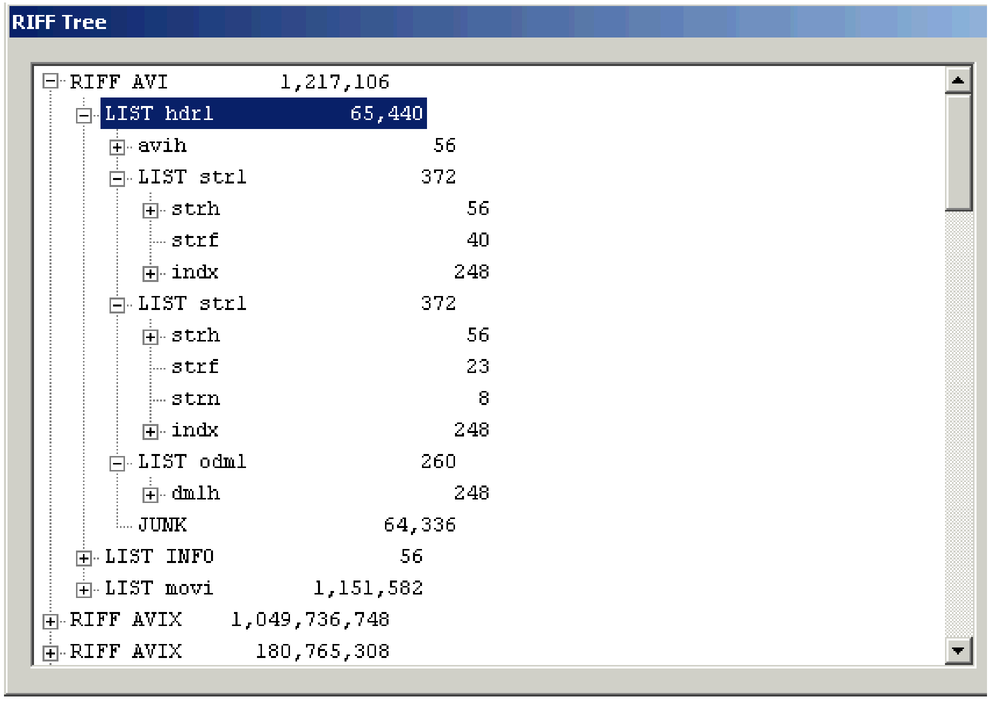
Unlike what a `uint32` suggests, the limit for the size of those lists is not 4 GB, but

- for AVI 1.0: size(RIFF-AVI) $<$ 2 GB
- for Open-DML:
  - `size(RIFF-AVI) < 1 GB` (!!) (assumed to be 2 GB by some muxing applications, like VirtualDub!)
  - `size(RIFF-AVIX) < 2 GB`

As Windows XP insists on reading the entire first RIFF AVI list if no Legacy Index (see page 12) is found, and as that Legacy Index causes overhead, it is recommended to create RIFF-AVI-Lists as small as possible.

2.1 Headers

The header section of an AVI file looks like this:



```
RIFF AVI          1,217,106
  LIST hdrl          65,440
    avih                 56
    LIST strl           372
      strh               56
      strf               40
      indx              248
    LIST strl           372
      strh               56
      strf               23
      strn                8
      indx              248
    LIST odml           260
      dmlh              248
    JUNK             64,336
  LIST INFO              56
  LIST movi       1,151,582
RIFF AVIX     1,049,736,748
RIFF AVIX       180,765,308
```

The following sections describe the meaning of these lists and chunks.

2.1.1 MainAVIHeader (avih)

This structure is defined as follows:

```
typedef struct
{
    DWORD dwMicroSecPerFrame; // frame display rate (or 0)
    DWORD dwMaxBytesPerSec; // max. transfer rate
    DWORD dwPaddingGranularity; // pad to multiples of this
                                                // size;
    DWORD dwFlags; // the ever-present flags
    DWORD dwTotalFrames; // # frames in file
    DWORD dwInitialFrames;
    DWORD dwStreams;
    DWORD dwSuggestedBufferSize;

    DWORD dwWidth;
    DWORD dwHeight;

    DWORD dwReserved[4];
} MainAVIHeader;
```

Unfortunately, those values do NOT have the meaning they seem to have when looking at their names.

- `dwMicroSecPerFrame`
  Contains the duration of one video frame in microseconds. This value can be ignored (see stream header), but shall be written correctly by any AVI writer.
  Important: Some broken programs, like *AVIFrate*, write the framerate value in the stream header, but not `dwMicroSecPerFrame`. Thus, `dwMicroSecPerFrame` should not be considered reliable!

- `dwMaxBytesPerSec`
  Highest occuring data rate within the file. That value is of no importance either. Its reliability should not be overrated.

- `dwPaddingGranularity`
  File is padded to a multiple of this

- `dwFlags`
  See below

- `dwTotalFrames`
  Contains the number of video frames in the RIFF-AVI list (it should NOT contain the total number of frames in the entire file if there are RIFF-AVIX-Lists. Some tools claiming to handle AVI files even assume this, but it definitely violates the Open-DML file format specification. Such applications are broken.) As some AVI file muxers write bad values here, this value should not be considered reliable.

- `dwInitialFrames`
  Ignore that

- `dwStreams`
  Number of streams in the file

- `dwSuggestedBufferSize`
  Size of buffer required to hold chunks of the file. The reliability of this value should not be overrated.

- `dwWidth`
  Width of video stream

- `dwHeight`
  Height of video stream

Available `Flags` for `MainAVIHeader::dwFlags`

- `AVIF_HASINDEX`
  The file has an index

- `AVIF_MUSTUSEINDEX`
  The order in which the video and audio chunks must be replayed is determined by the index and may differ from the order in which those chunks occur in the file.

- `AVIF_ISINTERLEAVED`
  The streams are properly interleaved into each other

- `AVIF_WASCAPTUREFILE`
  The file was captured. The interleave might be weird.

- `AVIF_COPYRIGHTED`
  Ignore it

- `AVIF_TRUSTCKTYPE` (Open-DML only!)
  This flag indicates that the keyframe flags in the index are reliable. If this flag is not set in an Open-DML file, the keyframe flags could be defective without technically rendering the file invalid.

2.1.2 The Stream header list - general

There is one `strl` - List for each stream. If the number of `strl` - Lists inside the `hdrl` - List is different from `MainAVIHeader::dwStreams`, a fatal error should be reported.

2.1.3 The stream header list element: strh

```
typedef struct {
    FOURCC  fccType;
    FOURCC  fccHandler;
    DWORD   dwFlags;
    WORD    wPriority;
    WORD    wLanguage;
    DWORD   dwInitialFrames;
    DWORD   dwScale;
    DWORD   dwRate;          /* dwRate / dwScale == samples/second */
    DWORD   dwStart;
    DWORD   dwLength;        /* In units above... */
    DWORD   dwSuggestedBufferSize;
    DWORD   dwQuality;
    DWORD   dwSampleSize;
    RECT    rcFrame;
} AVIStreamHeader;
```

Again, the meaning is not always obvious.

- `fccType`
  Can be
  - `'vids'` - video
  - `'auds'` - audio
  - `'txts'` - subtitle
- `fccHandler`
  FourCC of codec to be used.
- `dwFlags`
  The following flags are defined:
  - `AVISF_DISABLED` - Stream should not be activated by default
  - `AVISF_VIDEO_PALCHANGES` - Stream is a video stream using palettes where the palette is changing during playback.
- `dwInitialFrames`
  Number of the first block of the stream that is present in the file.
- `dwRate / dwScale` =
  `samples / second` (audio) or
  `frames / second` (video).
  dwScale and dwRate should be mutually prime. Tests have shown that for example 10,000,000/400,000 instead of 25/1 results in files that don't work on some hardware MPEG4 players.

- `dwStart`
  Start time of stream. In the case of VBR audio, this value indicates the number of silent frames to be played before the stream starts.
- `dwLength`
  size of stream in units as defined in `dwRate` and `dwScale`
- `dwSuggestedBufferSize`
  Size of buffer necessary to store blocks of that stream. Can be 0 (in that case the application has to guess), but should not be 0, as Microsoft's AVI splitter does not handle this case properly in some cases (e.g. MP3-CBR in Open-DML files)
- `dwQuality`
  should indicate the quality of the stream. Not important
- `dwSampleSize`
  number of bytes of one stream atom (that should not be split any further).

2.1.4 The stream header list element: strf

The structure of the `strf` chunk depends on the media type.
Video streams use the `BITMAPINFOHEADER` structure, whereas audio streams use the `WAVEFORMATEX` structure.

2.1.5 The stream header list element: indx

This chunk contains the upper level index for the stream. See page 13.

2.1.6 The stream header list element: strn

This element contains a name for the stream. That stream name should only use plain ASCII, especially not UTF-8.

3 AVI Indexes

3.1 old style index

The index as described is the index you will find in AVI 1.0 files. It is placed after the `movi` List in the `RIFF AVI List`. The data section of the idx1 chunk has the following layout:

```
AVIINDEXENTRY  index_entry[n]

typedef struct {
    DWORD  ckid;
    DWORD  dwFlags;
    DWORD  dwChunkOffset;
    DWORD  dwChunkLength;
} AVIINDEXENTRY;
```

Those values have the following meaning:

- **ckid**
  Specifies a four-character code corresponding to the chunk ID of a data chunk in the file.

- **dwFlags**
  The following flags are defined:

  - `AVIIF_KEYFRAME`: The chunk the entry refers to is a keyframe.
  - `AVIIF_LIST`: The entry points to a list, not to a chunk.
  - `AVIIF_FIRSTPART`: Indicates this chunk needs the frames following it to be used; it cannot stand alone.
  - `AVIIF_LASTPART`: Indicates this chunk needs the frames preceding it to be used; it cannot stand alone.
  - `AVIIF_NOTIME`: The duration which is applied to the corresponding chunk is 0.

  If neither `AVIIF_FIRSTPART` nor `AVIIF_LASTPART` is set, the chunk can be used alone, in other words, it is at least one packet of the corresponding stream. This is important for storing VBR audio streams in AVI files (see chapter 5.4)

- **dwChunkOffset**
  Contains the position of the header of the corresponding Chunk.
  Warning: This can be either the absolute position in the file, or the position relatively to the first byte of the `'movi'` identificator. An AVI File parser must be able to handle both versions.

- **dwChunkLength**
  Contains the size of the corresponding chunk in bytes.

3.2 Open-DML Index

The general structure of an Open-DML-Index-Chunk is the following:

```
typedef struct _aviindex_chunk {
  FOURCC fcc;
  DWORD  cb;
  WORD   wLongsPerEntry;
  BYTE   bIndexSubType;
  BYTE   bIndexType;
  DWORD  nEntriesInUse;
  DWORD  dwChunkId;
  DWORD  dwReserved[3];
  struct _aviindex_entry {
    DWORD  adw[wLongsPerEntry];
  } aIndex [ ];
} AVIINDEXCHUNK;
```

Every subtype of Open-DML index structures is compatible to this one. The elements have the following meaning:

- `fcc`, `cb`: Chunk header, same as `dwFourCC` and `dwSize` in the `CHUNK` structure
- `wLongsPerEntry`: every `aIndex[i]` has a size of 4*`wLongsPerEntry` bytes. (the structure of each `aIndex[i]` depends on the special type of index)
- `bIndexType`, `bIndexSubType`: defines the type of the index
- `nEntriesInUse`: `aIndex[0]..aIndex[nEntriesInUse-1]` are valid
- `dwChunkId`: ID of the stream the index points into, for example `'00dc'`.
  Consequently, one such index chunk can only point to data of one and the same stream.

3.2.1 Upper Level Index ('Super Index')

The upper level index ('super index') points to other index chunks and has the following structure:

```
typedef struct _avisuperindex_chunk {
  FOURCC fcc;
  DWORD  cb;
  WORD   wLongsPerEntry;
  BYTE   bIndexSubType;
  BYTE   bIndexType;
  DWORD  nEntriesInUse;
  DWORD  dwChunkId;
  DWORD  dwReserved[3];
  struct _avisuperindex_entry {
    __int64 qwOffset;
    DWORD   dwSize;
    DWORD   dwDuration;
  } aIndex[ ];
} AVISUPERINDEX;
```

The following values are now defined more specifically:

- `bIndexType = AVI_INDEX_OF_INDEXES`
- `bIndexSubType = [ AVI_INDEX_2FIELD | 0 ]`
- `wLongsPerEntry = 4`

As you can see, the `aIndex` array now consists of 4 `DWORD`s per entry. The values have the following meaning:

- `qwOffset`: Position of the index chunk this entry points to in the file
- `dwSize`: The size of the standard or field index chunk the entry is pointing to
- `dwDuration`: The duration, measured in stream ticks as indicated in the AVI stream header. In case of video or VBR audio, that usually refers to the number of frames.
  Important:
  VirtualDub 1.4.10 and earlier versions wrote b0rked values for this member in the audio stream. Thus, an AVI parser should be able to handle files without using this value!

3.2.2 The Standard Index

This index type contains pointers to video, audio or subtitle chunks. It also is a special form of the general Open-DML Index and looks like this:

```
  typedef struct _avistdindex_chunk {
    FOURCC  fcc;
    DWORD   cb;
    WORD    wLongsPerEntry;
    BYTE    bIndexSubType;
    BYTE    bIndexType;
    DWORD   nEntriesInUse;
    DWORD   dwChunkId;
    __int64 qwBaseOffset;
    DWORD dwReserved3;
    struct _avistdindex_entry {
      DWORD dwOffset;
      DWORD dwSize;
    } aIndex[ ];
} AVISTDINDEX;
```

- `wLongsPerEnrty`:
  As you can see easily, each `aIndex[i]` takes 8 bytes,
  so `wLongsPerEntry = 2`
- `bIndexSubType:  = 0`
- `bIndexType:= AVI_INDEX_OF_CHUNKS`
- `qwBaseOffset`:
  This value is added to each `dwOffset` value of the `AVISTDINDEX`.
- `dwOffset, dwSize`: These elements define the position (`qwBaseOffset + dwOffset`) of the data section of the corresponding `CHUNK` (NOT the chunk header!) and its length. There are `nEntriesInUse` such pairs, each one describing one video/audio frame. Note that Bit 31 of `dwSize` indicates the frametype: If this bit is set, this frame is not a keyframe.

Low-overhead mode

The Open-DML specification does not explicitly require that each "data section" the index contains an entry to be preceded by a chunk header. Thus, several frames can be put into one chunk, while having one index entry per frame. This way, only few frames have a chunk header, reducing the overhead by 50% compared to normal Open-DML files. This means, of course, that the `AVIF_MUSTUSEINDEX` flag in the main AVI header must be set, to force any parser to use the index. Files created this way will be called "low overhead AVI files".
This is not an AVI file type on its own. Any parser handling that flag, as well as the Open-DML index correctly, should be compatible to such files. Microsoft's AVI splitter, as well as VirtualDub(Mod) can handle such files without problems, without having been updated to do so.

3.3 Using the Open-DML Index

The preceding section described what the Open-DML Index looks like. This section will deal with using it.

Each stream contains an `'indx'` chunk in its stream header list (`'strl'`). This chunk is a Super Index chunk.

As each Standard Index contains one 64 bit offset and then a list of 32 bit offsets relatively to the 64 bit offset, one Standard Index chunk can only point to data within one 4 GB segment. Thus you need one Standard Index per 4 GB file size per stream.

Unfortunately, it seems that Microsoft did not read the specification properly: If a file contains more than 3 audio streams, then the Microsoft AVI Splitter will not recognize files using such large Standard Index chunks. It is required to use smaller pieces. Tests have shown that pieces with 15000 entries each are small enough to be processed correctly by Microsoft's AVI splitter.

4 The `movi` - Lists

The `Movi` - Lists contain Video, Audio, Subtitle and (secondary) index data. Those can be grouped into `rec ` - Lists. Example:

```
LIST movi
  LIST rec
    01wb
    01wb
    02wb
    03wb
    03wb
    03wb
    00dc
    00dc
  LIST rec
    01wb
    02wb
  LIST rec
    ...
    ...
    ix01
    ix02
    ix03
    ....
    ....
```

The following chunk header IDs are defined:

- ..wb: audio chunk
- ..dc: video chunk
- ..tx: subtitle chunk
- ix..: standard index block

Grouping chunks into `rec ` - Lists prevents excessive seeking when using the Microsoft AVI splitter for replay, but does not allow playback on some standalone replay devices.

The maximum size of a chunk of a stream should be smaller than the corresponding `dwSuggestedBufferSiz` value. Otherwise, some players, especially the Microsoft AVI splitter, could malfunction.

5 Audio types requiring special attention

5.1 MP3

`wFormatTag = 0x0055`

An MP3 audio stream consists of inseparable frames. MP3 decoders should be able to handle partial frames, but it is nevertheless recommended to store entire MP3 frames in the AVI chunks.

The `strf` chunk is an `MPEGLAYER3WAVEFORMAT` structure, which is an extention to the `WAVEFORMATEX` structure:

```
typedef struct mpeglayer3waveformat_tag {
  WAVEFORMATEX  wfx;
  WORD          wID;
  DWORD         fdwFlags;
  WORD          nBlockSize;
  WORD          nFramesPerBlock;
  WORD          nCodecDelay;
} MPEGLAYER3WAVEFORMAT;
```

Important:
This is only valid for MP3 ('MPEG Layer 3'), not for MP1 or MP2 ('MPEG Layer 1 / 2').

If the MP3 stream has a variable bitrate, then you need to convince DirectShow to seek properly. See section 5.4 (page 20) for more details on VBR audio streams in AVI files.

Unfortunately, whoever came up with the idea didn't think enough about it: It is possible to create MP3 audio frames larger than 1152 bytes if the sample rate is 32 khz or less. After reading and understanding section 5.4, you'll see why such audio frames render an MP3 stream unplayable if nBlockSize is set to 1152, which is usually done for MP3. Using a larger value would resolve this issue. However, some programs read an MP3 stream as VBR if and only if this value is exactly 1152. In other words, low sample rates in combination with high bitrates are a problem for MP3 VBR streams in AVI files.

5.2 AC3

`wFormatTag = 0x2000`

Muxing AC3 into AVI is far more problematic than most other audio formats. The reason is that a lot of decoders (software as well as hardware) are severely b0rked.

An AC3 stream consists, like MP3, of individual, inseparable frames. It is required that any audio chunk of an AVI file contains a few (complete!) AC3 frames. Otherwise, some AC3 decoders will miscalculate the duration of a chunk. As the audio stream is considered the master stream for playback in DirectShow, this miscalculation will lead to jerky video playback.

Theoretically, chunks containing one AC3 frame are valid, but there are hardware decoders which won't work with such streams. If you place more than 6 AC3 frames into one chunk, you might get increased playback speed. **Thus, the recommendation is to place between 2 and 5 AC3 frames into one AVI chunk**.

5.3 DTS

`wFormatTag = 0x2001`

It seems to work out to place between 2 and 20 DTS frames in one AVI chunk. I have not yet tried higher values on my own. If you have a hardware DTS decoder, please do some testing and report back what values work and which ones don't. Just like with AC3, do not split up any DTS frames.

5.4 VBR audio - general

Just like the video stream of virtually every AVI file does have a variable bitrate and a constant frame duration (e.g. 40ms), an audio stream of an AVI file can also have variable bitrate, if the frame duration is constant. This means, placing Vorbis audio in an AVI file will NOT result in anything playable using the method described below.

As the goal is to make DirectShow seek in the stream like in a video stream, it is required to use some of the stream header values in the same way they are used with video streams. That means:

- `dwRate` contains the sample rate
- `dwScale` contains the number of samples in one audio frame
- `WAVEFORMATEX::nBlockAlign` must have a value at least as big as the largest frame of that stream, measured in bytes.
- `dwSampleSize` = 0
- Each chunk contains entire stream packets and is therefore marked as such. Neither `AVIIF_LASTPART` nor `AVIIF_FIRSTPART` is set for any chunk. This way, DirectShow assumes each chunk to contain a packet of the duration indicated in the stream headers.

This way, you can ensure that the M$ AVI Splitter will believe that the duration of every chunk is
`[roundup(chunk size / nBlockAlign)] * dwScale / dwRate`

You can, of course, create units larger than one frame to decrease overhead (independantly from "low overhead muxing" as described on page 16).
However, if a stream is VBR, it is required that every chunk (in the case of "normal" files) or every piece of audio data the index points to (in the case of "low overhead AVI files") contains the same number of samples. The `dwScale` value needs to be set accordingly.

5.5 MPx VBR

- MPEG 1 Layer 3: 1152 samples per frame
- MPEG 2 Layer 3: 576 samples per frame
- MPEG 1/2 Layer 2: Strongly discouraged from being used, because the default MPEG Layer 2 decoder present on Windows systems does not recognise MPEG Layer 2 streams when using VBR headers.

5.6 AAC

For AAC, one RAW AAC frame usually spans over 1024 samples. However, depending on the source container (e.g. ADTS), it is theoretically possible that you are not able to extract packets of equal duration from your source file. In this case, it is highly recommended not to mux the AAC stream into AVI, but report a fatal error instead.

AAC and HE-AAC require private data in the corresponding `WAVEFORMATEX` structures. See the source code of AVI-Mux GUI (`FillASI.cpp`) for details.

5.7 VFR Audio - Storing Vorbis in AVI

Generally, the AVI file format does not support variable framerate streams. However, such streams can nevertheless be stored in AVI files when it is possible to create AVI chunks of constant duration, so that the stream can be treated as a normal VBR stream. There are basicly 2 possible ways to do that:

- Building larger "macro frames" of constant duration and placing one such macro frame into one chunk. This requires a decoder that is able to decode a sequence of independant frames at once, a decoder which does not require that only one single frame at once be transmitted. The currently available Vorbis decoders do no meet this requirement.
- The duration of one chunk is set to a very small value, several empty frames are added after one that is supposed to be long. This, however, causes a lot of overhead.

`FFMPEG` is using the second idea. However, ffmpeg is increasing the overhead even more, as it stores frame headers for each of the padding chunks, wasting 8 bytes each time, instead of only storing one of them physically, adding an index entry everytime such a chunk is required, and setting `AVIF_MUSTUSEINDEX`. For such streams, the following ID is used: `wFormatTag = 0x566F`. The Vorbis initialization packets (the first 3 packets of an OGG/Vorbis file) are stored as private data in the `strf` chunk. Each vorbis initialization packet is stored the following way:

```
big_endian_int16  size
char              data[size];
```

6 Subtitles in AVI files

This section explains how to store subtitles in AVI files, so that VSFilter can be used to load and select subtitles.

One subtitle stream is stored in one single chunk. That chunk contains header data, followed by an entire SRT or SSA file. If that file is using UTF-8 encoding, the BOM should be included as well. The header is defined as follows:

```
  char[4];  // 'GAB2'
  BYTE  0x00;
  WORD  0x02;              // unicode
  DWORD dwSize_name;       // length of stream name in bytes
  char  name[dwSize_name]; // zero-terminated subtitle stream
                              name encoded in UTF-16
  WORD  0x04;
  DWORD dwSize;            // size of SRT/SSA text file
  char  data[dwSize];      // entire SRT/SSA file
```

Stream header chunk

```
typedef struct {
    FOURCC  fccType;    // "txts"
    FOURCC  fccHandler; // 00 00 00 00
    DWORD   dwFlags;
    WORD    wPriority;
    WORD    wLanguage;
    DWORD   dwInitialFrames;
    DWORD   dwScale;
    DWORD   dwRate;       // dwRate / dwScale == duration in seconds
    DWORD   dwStart;
    DWORD   dwLength;     // In units above..., should be 1
    DWORD   dwSuggestedBufferSize;
    DWORD   dwQuality;
    DWORD   dwSampleSize; // = 0 -> treated as VBR
    RECT    rcFrame;      // 0, 0, 0, 0
} AVIStreamHeader;
```

Stream format chunk

This chunk has a size of 0.

Stream name chunk

The `strn` chunk is ignored by *VSfilter*, so there is no need to read or write it.

7 Garbage in AVI files

One way to set a delay in AVI files is using the `AVIStreamHeader::dwStart` value (see page 10). However, not every player reads this value, so that such files would not properly play on them. Some applications, like *VirtualDub* and derivates (*NanDub, VirtualDubMod*) add some data to the beginning of streams to be delayed.

Unfortunately, those applications don't care about the data format, but rather pad with zeros. Those zeros are refered to as 'garbage' in this section. The scope is to explain how to read such files, which requires that the duration of the garbage section be determined.

It is of course required that the beginning of the valid data be found, for example in the case of MP3, AC3 or DTS by looking for frame headers. Formats which don't use frame headers that can be detected easily, like AAC, will result in broken streams if zeros are added to the beginning.

7.1 Constant Bitrate

If the bitrate of a stream is constant, the duration of the garbage section is simply `garbage_length * data_rate`.

7.2 Variable Bitrate

In this case, the duration must be retrieved chunk-wise:

As described on page 20, the duration of one chunk is
`roundup(size_of_chunk / strf::nBlockAlign) * duration_per_frame`.

Consequently, if the beginning of the valid data is `the m-th byte` in chunk `n`, the duration of the garbage section is

```
sum[i=0..n-1](duration(chunk[i])) + roundup(m/nBlockAlign)
```

8 Overhead of AVI files

This section describes how to predict the overhead of an AVI file before muxing. Note: In the case of low overhead AVI files, the wording in this section is not applicable. Basicly, one video/audio frame causes about 8-9 bytes of overhead in low overhead AVI files.

8.1 General

The overhead of AVI files depends on the number of `CHUNK`s in the file. Other structures has only very little influence on the total overhead. Each `CHUNK` causes the following amount of overhead:

- 8 Bytes chunk header (all avi types)
- 16 Bytes for entry in Legacy Index (see page 12) (AVI 1.0 and the RIFF-AVI-List of Hybride files)
- 8 Bytes for entry in Standard Index (see page 15) (Open-DML)

That means, each `CHUNK` causes an amount of overhead of 16, 24 or 32 bytes.

8.2 Getting number of `CHUNKS`

8.2.1 Video

The easier part is the video stream: Each video frame takes one `CHUNK`.

8.2.2 Audio

The number of chunks for an audio stream depends on its format and packing. For specific formats, where very special packing is required or considered normal (see page 19), the overhead can be calculated easily from the settings. Otherwise, more precise information on the muxing settings is needed.

8.2.3 Examples

Video: 3 hours, 25 fps (= 3,600,000 / 40ms = 90,000 frames per hour)
Audio: 2x MP3-VBR (with 1 frame per `CHUNK` and 24 ms per frame)
Audio: 2x AC3 (with 4 frames per `CHUNK` and 32ms per frame)
-> Video: 270,000 `CHUNK`s
-> Audio: 2*150,000 + 2*3*28,125 = 468,750 `CHUNK`s
-> Sum = 738,750 `CHUNK`s