CHAPTER 1

AVI Files

The Microsoft Audio/Video Interleaved (AVI) file format is a RIFF file specification used with applications that capture, edit, and playback audio/video sequences. In general, AVI files contain multiple streams of different types of data. Most AVI sequences will use both audio and video streams. A simple variation for an AVI sequence uses video data and does not require an audio stream. Specialized AVI sequences might include a control track or MIDI track as an additional data stream. The control track could control external devices such as an MCI videodisc player. The MIDI track could play background music for the sequence. While a specialized sequence requires a specialized control program to take advantage of all its capabilities, applications that can read and play AVI sequences can still read and play an AVI sequence in a specialized file. (These applications ignore the non-AVI data in the specialized file.) This chapter primarily describes AVI files containing only audio and video data.

This chapter covers the following topics:

- The required chunks of an AVI file
- The optional chunks of an AVI file
- Developing routines to write AVI files

For additional information about RIFF files, see the *Microsoft Windows Multimedia Programmer's Guide* and *Microsoft Windows Multimedia Programmer's Reference*.

For additional information about installable compressors and decompressors, see chapter 10, "Video Compression and Decompression Drivers."

AVI RIFF Form

AVI files use the AVI RIFF form. The AVI RIFF form is identified by the four-character code "AVI ". All AVI files include two mandatory LIST chunks. These chunks define the format of the streams and stream data. AVI files might also include an index chunk. This

optional chunk specifies the location of data chunks within the file. An AVI file with these components has the following form:

```
RIFF ('AVI '
    LIST ('hdrl'
          .
          .
          .
        )
    LIST ('movi'
          .
          .
          .
        )
    ['idx1'<AVI Index>]
    )
```

The LIST chunks and the index chunk are subchunks of the RIFF “AVI ” chunk. The “AVI ” chunk identifies the file as an AVI RIFF file. The LIST “hdrl” chunk defines the format of the data and is the first required list chunk. The LIST “movi” chunk contains the data for the AVI sequence and is the second required list chunk. The “idx1” chunk is the optional index chunk. AVI files must keep these three components in the proper sequence.

The LIST “hdrl” and LIST “movi” chunks use subchunks for their data. The following example shows the AVI RIFF form expanded with the chunks needed to complete the LIST “hdrl” and LIST “movi” chunks:

```
RIFF ('AVI '
      LIST ('hdrl'
           'avih'(<Main AVI Header>)
           LIST ('strl'
                'strh'(<Stream header>)
                'strf'(<Stream format>)
                'strd'(additional header data)
                   .
                   .
                   .
           )


           .
           .
           .
  )


LIST ('movi'
           {SubChunk | LIST('rec '
                        SubChunk1
                        SubChunk2
                          .
                          .
                          .
                  )


              .
              .
              .
           }
      .
      .
      .
  )

  ['idx1'<AVIIndex>]
)
```

The following sections describe the chunks contained in the LIST “hdrl” and LIST “movi” chunks as well as the “idx1” chunk.

Data Structures for AVI Files

Data structures used in the RIFF chunks are defined in the AVIFMT.H header file. The reference section at the end of this chapter describes the data structures that can be used for the main AVI header, stream header, AVIIndex, and palette change chunks.

The Main AVI Header LIST

The file begins with the main header. In the AVI file, this header is identified with “avih” four-character code. The header contains general information about the file, such as the number of streams within the file and the width and height of the AVI sequence. The main header has the following data structure defined for it:

```
typedef struct {
    DWORD  dwMicroSecPerFrame;
    DWORD  dwMaxBytesPerSec;
    DWORD  dwReserved1;
    DWORD  dwFlags;
    DWORD  dwTotalFrames;
    DWORD  dwInitialFrames;
    DWORD  dwStreams;
    DWORD  dwSuggestedBufferSize;
    DWORD  dwWidth;
    DWORD  dwHeight;
    DWORD  dwScale;
    DWORD  dwRate;
    DWORD  dwStart;
    DWORD  dwLength;
} MainAVIHeader;
```

The **dwMicroSecPerFrame** field specifies the period between video frames. This value indicates the overall timing for the file.

The **dwMaxBytesPerSec** field specifies the approximate maximum data rate of the file. This value indicates the number of bytes per second the system must handle to present an AVI sequence as specified by the other parameters contained in the main header and stream header chunks.

The **dwFlags** field contains any flags for the file. The following flags are defined:

AVIF_HASINDEX

Indicates the AVI file has an “idx1” chunk.

AVIF_MUSTUSEINDEX

Indicates the index should be used to determine the order of presentation of the data.

AVIF_ISINTERLEAVED

Indicates the AVI file is interleaved.

AVIF_WASCAPTUREFILE

Indicates the AVI file is a specially allocated file used for capturing real-time video.

AVIF_COPYRIGHTED

Indicates the AVI file contains copyrighted data.

The AVIF_HASINDEX and AVIF_MUSTUSEINDEX flags applies to files with an index chunk. The AVI_HASINDEX flag indicates an index is present. The AVIF_MUSTUSEINDEX flag indicates the index should be used to determine the order of the presentation of the data. When this flag is set, it implies the physical ordering of the chunks in the file does not correspond to the presentation order.

The AVIF_ISINTERLEAVED flag indicates the AVI file has been interleaved. The system can stream interleaved data from a CD-ROM more efficiently than non-interleaved data. For more information on interleaved files, see “Special Information for Interleaved Files.”

The AVIF_WASCAPTUREFILE flag indicates the AVI file is a specially allocated file used for capturing real-time video. Typically, capture files have been defragmented by user so video capture data can be efficiently streamed into the file. If this flag is set, an application should warn the user before writing over the file with this flag.

The AVIF_COPYRIGHTED flag indicates the AVI file contains copyrighted data. When this flag is set, applications should not let users duplicate the file or the data in the file.

The **dwTotalFrames** field of the main header specifies the total number of frames of data in file.

The **dwInitialFrames** is used for interleaved files. If you are creating interleaved files, specify the number of frames in the file prior to the initial frame of the AVI sequence in this field.

The **dwStreams** field specifies the number of streams in the file. For example, a file with audio and video has 2 streams.

The **dwSuggestedBufferSize** field specifies the suggested buffer size for reading the file. Generally, this size should be large enough to contain the largest chunk in the file. If set to zero, or if it is too small, the playback software will have to reallocate memory during playback which will reduce performance. For an interleaved file, the buffer size should be large enough to read an entire record and not just a chunk.

The **dwWidth** and **dwHeight** fields specify the width and height of the AVI file in pixels.

The **dwScale** and **dwRate** fields are used to specify the general time scale that the file will use. In addition to this time scale, each stream can have its own time scale. The time scale in samples per second is determined by dividing **dwRate** by **dwScale**.

The **dwStart** and **dwLength** fields specify the starting time of the AVI file and the length of the file. The units are defined by **dwRate** and **dwScale**. The **dwStart** field is usually set to zero.

The Stream Header (“strl”) Chunks

The main header is followed by one or more “strl” chunks. (A “strl” chunk is required for each data stream.) These chunks contain information about the streams in the file. Each “strl” chunk must contain a stream header and stream format chunk. Stream header chunks are identified by the four-character code “strh” and stream format chunks are identified with the four-character code “strf”. In addition to the stream header and stream format chunks, the “strl” chunk might also contain a stream data chunk. Stream data chunks are identified with the four-character code “strd”.

The stream header has the following data structure defined for it:

```
typedef struct {
   FOURCC  fccType;
   FOURCC  fccHandler;
   DWORD   dwFlags;
   DWORD   dwReserved1;
   DWORD   dwInitialFrames;
   DWORD   dwScale;
   DWORD   dwRate;
   DWORD   dwStart;
   DWORD   dwLength;
   DWORD   dwSuggestedBufferSize;
   DWORD   dwQuality;
   DWORD   dwSampleSize;
} AVIStreamHeader;
```

The stream header specifies the type of data the stream contains, such as audio or video, by means of a four-character code. The **fccType** field is set to “vids” if the stream it specifies contains video data. It is set to “auds” if it contains audio data.

The **fccHandler** field contains a four-character code describing the installable compressor or decompressor used with the data.

The **dwFlags** field contains any flags for the data stream. The AVISF_DISABLED flag indicates that the stream data should be rendered only when explicitly enabled by the user. The AVISF_VIDEO_PALCHANGES flag indicates palette changes are embedded in the file.

The **dwInitialFrames** is used for interleaved files. If you are creating interleaved files, specify the number of frames in the file prior to the initial frame of the AVI sequence in this field.

The remaining fields describe the playback characteristics of the stream. These factors include the playback rate (**dwScale** and **dwRate**), the starting time of the sequence (**dwStart**), the length of the sequence (**dwLength**), the size of the playback buffer (**dwSuggestedBuffer**), an indicator of the data quality (**dwQuality**), and sample size (**dwSampleSize**). See the reference section for more information on these fields.

Some of the fields in the stream header structure are also present in the main header structure. The data in the main header structure applies to the whole file while the data in the stream header structure applies only to a stream.

A stream format (“strf”) chunk must follow a stream header (“strh”) chunk. The stream format chunk describes the format of the data in the stream. For video streams, the information in this chunk is a BITMAPINFO structure (including palette information if appropriate). For audio streams, the information in this chunk is a WAVEFORMATEX or PCMWAVEFORMAT structure. (The WAVEFORMATEX structure is an extended version of the WAVEFORMAT structure.) For more information on this structure, see the *New Multimedia Data Types and Data Techniques Standards Update*.

The “strl” chunk might also contain a stream data (“strd”) chunk. If used, this chunk follows the stream format chunk. The format and content of this chunk is defined by installable compression or decompression drivers. Typically, drivers use this information for configuration. Applications that read and write RIFF files do not need to decode this information. They transfer this data to and from a driver as a memory block.

An AVI player associates the stream headers in the LIST “hdrl” chunk with the stream data in the LIST “movi” chunk by using the order of the “strl” chunks. The first “strl” chunk applies to stream 0, the second applies to stream 1, and so forth. For example, if the first “strl” chunk describes the wave audio data, the wave audio data is contained in stream 0. Similarly, if the second “strl” chunk describes video data, then the video data is contained in stream 1.

The LIST “movi” Chunk

Following the header information is a LIST “movi” chunk that contains chunks of the actual data in the streams; that is, the pictures and sounds themselves. The data chunks can reside directly in the LIST “movi” chunk or they might be grouped into “rec ” chunks. The “rec ” grouping implies that the grouped chunks should be read from disk all at once. This is used only for files specifically interleaved to play from CD-ROM.

Like any RIFF chunk, the data chunks contain a four-character code to identify the chunk type. The four-character code that identifies each chunk consists of the stream number and a two-character code that defines the type of information encapsulated in the chunk. For example, a waveform chunk is identified by a two-character code of “wb”. If a waveform chunk corresponded to the second LIST “hdrl” stream description, it would have a four-character code of “01wb”.

Since all the format information is in the header, the audio data contained in these data chunks does not contain any information about its format. An audio data chunk has the following format (the ## in the format represents the stream identifier):

```
WAVE  Bytes   '##wb'
     BYTE    abBytes[];
```

Video data can be compressed or uncompressed DIBs. An uncompressed DIB has BI_RGB specified for the **biCompression** field in its associated BITMAPINFO structure. A compressed DIB has a value other than BI_RGB specified in the **biCompression** field. For more information about compression formats, see the description of the BITMAPINFOHEADER data structure in the *Microsoft Windows Programmers Reference* and Chapter 5, “DIB Format Extensions for Microsoft Windows.”

A data chunk for an uncompressed DIB contains RGB video data. These chunks are identified with a two-character code of “db” (db is an abbreviation for DIB bits). Data chunks for a compressed DIB are identified with a two-character code of “dc” (dc is an abbreviation for DIB compressed). Neither data chunk will contain any header information about the DIBs. The data chunk for an uncompressed DIB has the following form:

```
DIB  Bits   '##db'
    BYTE   abBits[];
```

The data chunk for a compressed DIB has the following form:

```
Compressed DIB   '##dc'
    BYTE        abBits[];
```

Video data chunks can also define new palette entries used to update the palette during an AVI sequence. These chunks are identified with a two-character code of “pc” (pc is an abbreviation for palette change). The following data structure is defined palette information:

```
typedef struct {
   BYTE         bFirstEntry;
   BYTE         bNumEntries;
   WORD          wFlags;
   PALETTEENTRY  peNew;
} AVIPALCHANGE;
```

The **bFirstEntry** field defines the first entry to change and the **bNumEntries** field specifies the number of entries to change. The **peNew** field contains the new color entries.

If you include palette changes in a video stream, set the AVITF_VIDEO_PALCHANGES flag in the **dwFlags** field of the stream header. This flag indicates that this video stream contains palette changes and warns the playback software that it will need to animate the palette.

The “idx1” Chunk

AVI files can have an index chunk after the LIST “movi” chunk. The index chunk essentially contains a list of the data chunks and their location in the file. This provides efficient random access to the data within the file, because an application can locate a particular sound sequence or video image in a large AVI file without having to scan it.

Index chunks use the four-character code “idx1”. The following data structure is defined for index entries:

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

The **ckid**, **dwFlags**, **dwChunkOffset**, and **dwChunkLength** entries are repeated in the AVI file for each data chunk indexed. If the file is interleaved, the index will also have these entries for each “rec” chunk. The “rec” entries should have the AVIIF_LIST flag set and the list type in the **ckid** field.

The **ckid** field identifies the data chunk. This field uses four-character codes for identifying the chunk.

The **dwFlags** field specifies any flags for the data. The AVIIF_KEYFRAME flag indicates key frames in the video sequence. Key frames do not need previous video information to be decompressed. The AVIIF_NOTIME flag indicates a chunk does not affect the timing of a video stream. For example, changing palette entries indicated by a palette chunk should occur between displaying video frames. Thus, if an application needs to determine the length of a video sequence, it should not use chunks with the AVIIF_NOTIME flag. In this case, it would ignore a palette chunk. The AVIIF_LIST flag indicates the current chunk is a LIST chunk. Use the **ckid** field to identify the type of LIST chunk.

The **dwChunkOffset** and **dwChunkLength** fields specify the position of the chunk and the length of the chunk. The **dwChunkOffset** field specifies the position of the chunk in the file relative to the 'movi' list. The **dwChunkLength** field specifies the length of the chunk excluding the eight bytes for the RIFF header.

If you include an index in the RIFF file, set the AVIF_HASINDEX in the **dwFlags** field of the AVI header. (This header is identified by “avih” chunk ID.) This flag indicates that the file has an index.

Other Data Chunks

If you need to align data in your AVI file you can add a “JUNK” chunk. (This chunk is a standard RIFF type.) Applications reading these chunks ignore their contents. Files played from CD-ROM use these chunks to align data so they can be read more efficiently. You might want to use this chunk to align your data for the 2 kilobyte CD-ROM boundaries. The “JUNK” chunk has the following form:

```
AVI Padding   'JUNK'
   Byte      data[]
```

As with any other RIFF files, all applications that read AVI files should ignore the non-AVI chunks that it does not recognize. Applications that read and write AVI files should preserve the non-AVI chunks when they save files they have loaded.

Special Information for Interleaved Files

Files that are interleaved for playback from CD-ROM require some special handling. While they can be read similarly to any other AVI files, they require special care when produced.

The audio has to be separated into single-frame pieces, and audio and video for each frame needs to be grouped together into “rec ” chunks. The record chunks should be padded so that their size is a multiple of 2 kilobytes and so that the beginning of the actual data in the LIST chunk lies on a 2 kilobyte boundary in the file. (This implies that the LIST chunk itself begins 12 bytes before a 2 kilobyte boundary.)

To give the audio driver enough audio to work with, the audio data has to be skewed from the video data. Typically, the audio data should be moved forward enough frames to allow approximately 0.75 seconds of audio data to be preloaded. The **dwInitialRecords** field of the main header and the **dwInitialFrames** field of the audio stream header should be set to the number of frames the audio is skewed.

Additionally, you must ensure that CD-ROM drive is capable of reading the data fast enough to support your AVI sequence. Non-MPC CD-ROM drives can have a data rate of less than 150 kilobytes per second.

Using VidEdit With AVI Files

VidEdit lets you create and edit audio-visual sequences consisting of a series of frames that contain digital audio and video data. You can use VidEdit to create and edit AVI files that contain one audio and one video stream. Each stream in the file must start at the beginning of the file (that is, the **dwStart** field in each stream header must be zero).

Example Code for Writing AVI Files

The WRITEAVI.C and AVIEASY.C files contain example code for writing AVI files. For simplicity, the examples assume that all video frames are uncompressed DIBs of the same size. While the DIBS can have any bit depth; 8, 16, and 24 bits are preferred.

These examples also assume all wave data is in memory. A more generalized procedure should work with wave data that is in memory as well as in a disk file. These examples do not restrict wave data to PCM. It should work with any format.

An Outline for Writing AVI Files

Like other RIFF files, AVI files are created with the **mmioOpen**, **mmioCreateChunk**, **mmioWrite**, **mmioAscend**, and **mmioClose** functions. These functions have the following definitions:

mmioOpen	Opens a file for reading or writing, and returns a handle to the open file.
mmioCreateChunk	Creates a new chunk in a RIFF file.
mmioWrite	Writes a specified number of bytes to an open file.
mmioAscend	Ascends out of a RIFF file chunk to the next chunk in the file.
mmioClose	Closes an open file.

In addition to these functions, you can use **mmioFOURCC** to convert four individual characters into a four-character code. For more information on these functions and macros, see the *Microsoft Windows Multimedia Programmer's Guide* and *Microsoft Windows Multimedia Programmer's Reference*.

The AVIFMT.H file contains macro definitions for creating the two- and four-character codes described in this chapter. It also defines the **aviTWOCC** and **TWOCCFromFOURCC** macros. These macros create two-character codes from individual characters or from four-character codes.

Unlike many other RIFF files, AVI files use many nested chunks and subchunks. This makes them more complicated than most RIFF files. Use the following tables as a checklist to help you decide when to create a chunk, when to write data to a chunk, and when to ascend from a chunk. The tables do not include information about writing non-AVI data chunks to the file. The information in the chunk column of the table mirrors the example in the "AVI RIFF Form" section presented previously.

Creating the File and "AVI " Chunk

The "AVI " chunk is the first chunk in the file. You will not ascend from this chunk until all other chunks have been created.

Chunk	How to Handle
RIFF ('AVI '	Use **mmioOpen** to open the file. Seek to the beginning of the file with **mmioSeek**. Create the AVI chunk with **mmioCreateChunk**. (Use the “AVI ” four-character code and the MMIO_CREATERIFF flag.) Do not ascend from this chunk in preparation for writing the remaining chunks.

Creating the LIST “hdrl ” and “avih” Chunks

The LIST “hdrl ” chunk contains the stream format header chunks. Because it contains other chunks, you will not ascend from it until the other header chunks are created.

The “avih” chunk contains the main header list. This is written as a complete chunk.

Chunk	How to Handle
LIST ('hdrl'	Create the LIST “hdrl” chunk with **mmioCreateChunk**. (Use the “hdrl” four-character code and the MMIO_CREATELIST flag.)
'avih'(<Main AVI Header>)	Create the Main AVI Header chunk with **mmioCreateChunk**. (Use the “avih” four-character code.) Write the header information with **mmioWrite**. Ascend from the “avih” chunk with **mmioAscend**. Do not ascend from the LIST “hdrl” chunk.

Creating the “strl”, “strh”, “strf”, and “strd” Chunks

The “strl”, “strh”, “strf”, and “strd” chunks are written as complete chunks. You write a set of the “strh”, “strf”, and “strd” chunks for each stream in the file. After all the stream descriptions are written, you ascend from LIST “hdrl” chunk.

Chunk	How to Handle
LIST ('strl'	Create the LIST “strl” chunk with **mmioCreateChunk**. (Use the “strl” four-character code and the MMIO_CREATELIST flag.)
'strh'(<Stream header>)	Create the stream header chunk with **mmioCreateChunk**. (Use the “strh” four-character code.) Write the stream header information with **mmioWrite**. Ascend from the “strh” chunk with **mmioAscend**.
'strf'(<Stream format>)	Create the stream format chunk with **mmioCreateChunk**. (Use the “strf” four-character code.) Write the stream format information with **mmioWrite**. Ascend from the “strf” chunk with **mmioAscend**.
'strd'(additional header data)	If needed, create chunks for any additional header data with **mmioCreateChunk**. (Use the “strd” four-character code.) Write the additional header data with **mmioWrite**. Ascend from the “strd” chunk.
. . .	If needed, add stream header, stream format, and additional header data chunks for other streams in the file.
)	Ascend from the LIST “strl” chunk with **mmioAscend**.

. . .)	Ascend from the LIST “hdrl” chunk with **mmioAscend**. If needed, create and write padding chunks or other data chunks.

Creating the LIST “movi” and “rec ” Chunks

The LIST “movi” chunk contains other chunks. After you create this chunk, you will not ascend from it until the other chunks are written.

You can write the data as an individual chunk or as part of a “rec ” chunk. Like the LIST “movi” chunk, you will not ascend from a “rec ” chunk until you write all of its subchunks.

Chunk	**How to Handle**
LIST ('movi'	Create the LIST “movi” chunk with **mmioCreateChunk**. (Use the LIST “movi” four-character code and the MMIO_CREATELIST flag.)

{ SubChunk |
    LIST('rec '
        SubChunk1
        SubChunk2
         .
         .
         .
)
    .
    .
    .
    }
	You can add your movie data directly at this point in a subchunk or include it in a “rec ” chunk. The following steps summarize creating these chunks: Create a data chunk with **mmioCreateChunk**. (Use the four-character code appropriate for the data chunk and stream.) If you are adding an index chunk to the end of the file, save the location of the subchunks for it.

.	
.	
.	
)	Ascend from the LIST “movi ” chunk.

Creating the “idx1” Chunk and Ascending From the “AVI ” Chunk

The optional index chunk is written as a complete chunk. After you have completed this chunk, you can ascend from the “AVI ” chunk and close the file.

Chunk	**How to Handle**
['idx1'<AVIIndex>]	If used, create the AVI index chunk with **mmioCreateChunk**. (Use the “idx1” four-character code.) Write the index information with **mmioWrite**. Ascend from the “idx1” chunk with **mmioAscend**. Although the “idx1” is the last chunk used in an AVI sequence, you can add non-AVI chunks after it. These subchunks will still be part of the “AVI ” chunk.
)	Ascend from the “AVI ” chunk with **mmioAscend**. Close the file with **mmioClose**.

AVI RIFF File Reference

This section lists data structures used to support AVI RIFF files. (These structures are defined in AVIFMT.H.) The data structures are presented in alphabetical order. The structure definition is given, followed by a description of each field.

AVIINDEXENTRY

The AVI file index consists of an array of **AVIINDEXENTRY** structures contained within an 'idx1' chunk at the end of an AVI file. This chunk follows the main LIST 'movi' chunk which contains the actual data.

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

Fields

The **AVIINDEXENTRY** structure has the following fields:

ckid

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

dwFlags

Specifies any applicable flags. The flags in the low-order word are reserved for AVI, while those in the high-order word can be used for stream- and compressor/decompressor-specific information.

The following values are currently defined:

AVIIF_LIST
Indicates the specified chunk is a 'LIST' chunk, and the **ckid** field contains the list type of the chunk.

AVIIF_KEYFRAME
Indicates this chunk is a key frame. Key frames do not require additional preceding chunks to be properly decoded.

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

Indicates this chunk should have no effect on timing or calculating time values based on the number of chunks. For example, palette change chunks in a video stream should have this flag set, so that they are not counted as taking up a frame’s worth of time.

dwChunkOffset

Specifies the position in the file of the specified chunk. The position value includes the eight byte RIFF header.

dwChunkLength

Specifies the length of the specified chunk. The length value does not include the eight byte RIFF header.

AVIPALCHANGE

The **AVIPALCHANGE** structure is used in video streams containing palettized data to indicate the palette should change for subsequent video data.

```
typedef struct {
   BYTE  bFirstEntry;
   BYTE  bNumEntries;
   WORD  wFlags;
   PALETTEENTRY  peNew;
} AVIPALCHANGE;
```

Fields

The **AVIPALCHANGE** structure has the following fields:

bFirstEntry

Specifies the first palette entry to change.

bNumEntries

Specifies the number of entries to change.

wFlags

Reserved. (This should be set to 0.)

peNew

Specifies an array of new palette entries.

AVIStreamHeader

The **AVIStreamHeader** structure contains header information for a single stream of an file. It is contained within an 'strh' chunk within a LIST 'strl' chunk that is itself contained within the LIST 'hdrl' chunk at the beginning of an AVI RIFF file.

```
typedef struct {
   FOURCC  fccType;
   FOURCC  fccHandler;
   DWORD  dwFlags;
   DWORD  dwReserved1;
   DWORD  dwInitialFrames;
   DWORD  dwScale;
   DWORD  dwRate;
   DWORD  dwStart;
   DWORD  dwLength;
   DWORD  dwSuggestedBufferSize;
   DWORD  dwQuality;
   DWORD  dwSampleSize;
} AVIStreamHeader;
```

Fields

The **AVIStreamHeader** structure has the following fields:

fccType

Contains a four-character code which specifies the type of data contained in the stream. The following values are currently defined for AVI data:

'vids'
Indicates the stream contains video data. The stream format chunk contains a **BITMAPINFO** structure which can include palette information.

'auds'
Indicates the stream contains video data. The stream format chunk contains a **WAVEFORMAT** or **PCMWAVEFORMAT** structure.

Other four-character codes can identify non-AVI data.

fccHandler

Optionally, contains a four-character code that identifies a specific data handler. The data handler is the preferred handler for the stream.

dwFlags

Specifies any applicable flags. The bits in the high-order word of these flags are specific to the type of data contained in the stream. The following flags are currently defined:

AVISF_DISABLED
Indicates this stream should not be enabled by default.

AVISF_VIDEO_PALCHANGES

Indicates this video stream contains palette changes. This flag warns the playback software that it will need to animate the palette.

dwReserved1

Reserved. (Should be set to 0.)

dwInitialFrames

Specifies how far audio data is skewed ahead of the video frames in interleaved files. Typically, this is about 0.75 seconds.

dwScale

This field is used together with **dwRate** to specify the time scale that this stream will use.

Dividing **dwRate** by **dwScale** gives the number of samples per second.

For video streams, this rate should be the frame rate.

For audio streams, this rate should correspond to the time needed for **nBlockAlign** bytes of audio, which for PCM audio simply reduces to the sample rate.

dwRate

See **dwScale**.

dwStart

Specifies the starting time of the AVI file. The units are defined by the **dwRate** and **dwScale** fields in the main file header. Normally, this is zero, but it can specify a delay time for a stream which does not start concurrently with the file.

Note: The 1.0 release of the AVI tools does not support a non-zero starting time.

dwLength

Specifies the length of this stream. The units are defined by the **dwRate** and **dwScale** fields of the stream's header.

dwSuggestedBufferSize

Suggests how large a buffer should be used to read this stream. Typically, this contains a value corresponding to the largest chunk present in the stream. Using the correct buffer size makes playback more efficient. Use zero if you do not know the correct buffer size.

dwQuality

Specifies an indicator of the quality of the data in the stream. Quality is represented as a number between 0 and 10000. For compressed data, this typically represent the value of the quality parameter passed to the compression software. If set to -1, drivers use the default quality value.

dwSampleSize

Specifies the size of a single sample of data. This is set to zero if the samples can vary in size. If this number is non-zero, then multiple samples of data can be grouped into a single chunk within the file. If it is zero, each sample of data (such as a video frame) must be in a separate chunk.

For video streams, this number is typically zero, although it can be non-zero if all video frames are the same size.

For audio streams, this number should be the same as the **nBlockAlign** field of the **WAVEFORMAT** structure describing the audio.

MainAVIHeader

The **MainAVIHeader** structure contains global information for the entire AVI file. It is contained within an 'avih' chunk within the LIST 'hdrl' chunk at the beginning of an AVI RIFF file.

```
typedef struct {
   DWORD  dwMicroSecPerFrame;
   DWORD  dwMaxBytesPerSec;
   DWORD  dwReserved1;
   DWORD  dwFlags;
   DWORD  dwTotalFrames;
   DWORD  dwInitialFrames;
   DWORD  dwStreams;
   DWORD  dwSuggestedBufferSize;
   DWORD  dwWidth;
   DWORD  dwHeight;
   DWORD  dwScale;
   DWORD  dwRate;
   DWORD  dwStart;
   DWORD  dwLength;
} MainAVIHeader;
```

Fields

The **MainAVIHeader** structure has the following fields:

dwMicroSecPerFrame

Specifies the number of microseconds between frames.

dwMaxBytesPerSec

Specifies the approximate maximum data rate of file.

dwReserved1

Reserved. (This field should be set to 0.)

dwFlags

Specifies any applicable flags. The following flags are defined:

AVIF_HASINDEX
Indicates the AVI file has an 'idx1' chunk containing an index at the end of the file. For good performance, all AVI files should contain an index.

AVIF_MUSTUSEINDEX
Indicates that the index, rather than the physical ordering of the chunks in the file, should be used to determine the order of presentation of the data. For example, this could be used for creating a list frames for editing.

AVIF_ISINTERLEAVED
Indicates the AVI file is interleaved.

AVIF_WASCAPTUREFILE

Indicates the AVI file is a specially allocated file used for capturing real-time video. Applications should warn the user before writing over a file with this flag set because the user probably defragmented this file.

AVIF_COPYRIGHTED

Indicates the AVI file contains copyrighted data and software. When this flag is used, software should not permit the data to be duplicated.

dwTotalFrames

Specifies the number of frames of data in file.

dwInitialFrames

Specifies the initial frame for interleaved files. Non-interleaved files should specify zero.

dwStreams

Specifies the number of streams in the file. For example, a file with audio and video has 2 streams.

dwSuggestedBufferSize

Specifies the suggested buffer size for reading the file. Generally, this size should be large enough to contain the largest chunk in the file. If set to zero, or if it is too small, the playback software will have to reallocate memory during playback which will reduce performance.

For an interleaved file, this buffer size should be large enough to read an entire record and not just a chunk.

dwWidth

Specifies the width of the AVI file in pixels.

dwHeight

Specifies the height of the AVI file in pixels.

dwScale

This field is used with **dwRate** to specify the time scale that the file as a whole will use. In addition, each stream can have its own time scale.

Dividing **dwRate** by **dwScale** gives the number of samples per second.

dwRate

See **dwScale**.

dwStart

Specifies the starting time of the AVI file. The units are defined by **dwRate** and **dwScale**. This field is usually set to zero.

dwLength

Specifies the length of the AVI file. The units are defined by **dwRate** and **dwScale**. This length is returned by MCIAVI when using the frames time format.