Appendix F
DVI– Multimedia File Format

xe "DVI multimedia file format F-"§

This appendix describes the DVI– multimedia file format used for motion video and audio objects, and for storing compressed and uncompressed still images.

This appendix provides information about:

- File structure and version control strategy
- File organization, including tables that describe fields, their settings, and detailed comments, as appropriate
- Derived values

In addition, because of its compatibility with files produced using the ActionMedia– II software, the old still image format is documented in this appendix.

Introduction

DVI technology has defined a file format for storing audio/video objects. Applications should use this and other industry standard file formats, to increase interoperability with other applications such as media editing and manipulation tools. The DVI multimedia file format is particularly appropriate for motion

video objects that use the compression algorithms, and media objects that use ActionMedia II board pixel formats.

Although this appendix references the routines and concepts used with AVK, the file format itself is not restricted for use only with AVK.

The DVI multimedia file format was designed to grow into a general purpose repository for complex multimedia objects, including information that might be added by media object editors. Therefore, it has some reserved fields that are not needed for playback of existing files.

The AVK specification does not require the use of the DVI multimedia file format. However, AVK only supports the data streaming file conventions used in DVI multimedia files. Applications can use other data streaming conventions, by converting the data before passing it to and from group buffers. The AVKIO sample programs in the AVK software release provide examples of programs that read and write files in this format.

General Considerations

Each audio or video file contains one or more streams of data. The following information applies to streams and stream data.

- Each stream typically contains digital data that describes a single audio or video stream. For example, an audio stream can contain ADPCM4 encoded data describing a waveform audio channel. Generally, there can be several such streams, all of which are intended for

simultaneous playback.

- To reduce head movement on the storage device, the data from the various streams must be interleaved. A frame is the unit of interleaving, and is nominally 1/30 of a second. The actual duration of a frame is stored within the file header.

The following sections specify and describe recommended field values to use under AVK. Some existing ActionMedia II software files might use values different from those recommended in this appendix.

The descriptions list fields, specify how these fields should be referenced when playing a file, and describe fields that can be ignored. To explain how fields should be referenced, the following criteria are used:

- Some fields are used only to verify that the file conforms to a variation of the file structure that can be played by AVK.
- Some fields are used to access information that must be passed to AVK through appropriate API routines.
- Some fields can be safely ignored.

File Structure

xe "File structure F-"§

A file consists of a set of inter-related data structures that describe the organization of the data into streams, the nature of the data in each stream, and the actual data itself. The following information describes the various fields in these data structures, and how to generate and interpret the data they contain.

Some of these fields are shown as bytes, words, or longs (U8, I16, U16, I32 or U32). These are standard AVK data types defined in AVKCOM.H. In the context of file interchange it is especially important that the precise length of integer fields be specified unambiguously. In many cases, this appendix gives both the symbolic names that are defined in .H include files provided with the AVK product, and the current values associated with these symbols.

The data structures also include explicit fields whose primary purpose is to force compiler-independent word and long alignment, as appropriate. These fields use little endian byte ordering. If these files are used with other processor hardware, the associated software must convert to and from the corresponding byte order.

Some of the fields are described as being "offsets". In this context, an offset is a byte count, measured from the beginning of the file to the first byte of some data in the file. While this appendix sometimes specifies recommended values for these offsets, during playback the actual offset in the file should always be used. Otherwise, there might be difficulty processing existing files and future extensions to the file format. Generally, software can move the data pointed at by an offset elsewhere in the file, simply by changing the offset value.

As a general rule, an offset of zero means that the associated data is not present.

Version Control Strategy

xe "File structure: Version control strategy F-"§

The data structures within this file format use a common strategy to allow controlled growth in functionality, without breaking previous functionality. This strategy is implemented by the use of three fields at the beginning of each file and the beginning of many internal data structures. These fields are a four character ASCII ID, a version number and a size in bytes. All three fields are useful, since each deals with a different kind of version mismatch, or binding mismatch problem.

HdrID Field

xe "Version control strategy:HdrID field F-"§

The *HdrID* field is used to validate that this structure is the expected kind of data structure. For the *HdrID* field at the beginning of the file, such a validation is essential, since the host file system allows end users to move and rename files at will. For the *HdrID* fields in internal data structures, validation of this field merely provides some assurance that the file data has not been corrupted.

HdrSize Field

xe "Version control strategy:HdrSize field F-"§

The *HdrSize* field gives the length of the data structure in bytes, and is central to the file version control strategy. The file format is modified by adding new fields at the end of a data structure. Software that uses the latest version of the file format must properly set all fields when it creates a file.

Fields that are set include those fields that are no longer needed by the latest version of file-reading software. Setting fields needed by previous versions allows older programs that have not been upgraded to the latest file format version to operate correctly within the limitations of the older file format version. In addition, since some data might not have existed when a software version was compiled, extensions to the file format have been carefully limited in ways that prevent old software from misinterpreting data.

File-reading software deals with expected values in three ways:

1. If the *HdrSize* in the file is the expected value, this data structure has the expected format. In this case, application software can safely interpret the fields, as described in the section, "File Organization".
2. If the *HdrSize* in the file is less than the expected value, then this is an old format file, and is missing some expected information. Each file format version contains enough information for the level of processing that had been supported at the time the file was created. In this case, no missing information was essential to processing the file(that is, the fields missing from the file contain clearly-defined default values that can be used instead of the missing values).

For example, the *AvLCim.DCFId* field was not part of the original file format definition. This field is set by a Digital Compression Facility (DCF) to provide information on where the video was compressed. For files that do not contain this field, as indicated by *AvLCim.HdrSize* (or *AvLCim.HdrVer*), the default value specifies that the compression site is unknown.

A convenient software technique for dealing with the possibility that *HdrSize* is less than the expected value is to initialize a copy of the data structure with default values, and then only read in *HdrSize* bytes as given by the actual *HdrSize* field in the file.

3. If the *HdrSize* in the file is greater than the expected value, then this file has a format which was extended after the code was written. In this case, there are new fields that have been defined, but the application code lacks the knowledge to interpret them.

   For file-reading, therefore, only use the information that is described in the version of the file format definition that existed when the code was written. When an old executable is provided new format files to process, the executable might be able to play or process the new format files by ignoring fields that did not exist when the old executable was compiled.

To support this processing scenario, applications always set all fields with appropriate values. These fields can be used by older versions of the software, but are ignored by the latest version of the software.

A desirable file format extension might require the addition of new fields that might produce files that could not be properly played by old executables. In this case, a new type or SubType might be introduced, as discussed in the section called, "Type and Subtype Fields". Use of new Types and Subtypes makes the new data invisible to the old executable.

Although in some situations data can be transcribed from an old file to a new file, even without knowing what data is represented in certain fields, it is recommended that applications totally ignore data in unknown Type or Subtype fields. Ignoring the data is the only guaranteed way to produce software that is compatible with a later file format definition.

HdrVersion Field

xe "Version control strategy:HdrVersion field F-"§

The corresponding *HdrVersion* field is incremented whenever a new software release adds new fields to a file data structure. Thus, either the *HdrVersion* or *HdrSize* field can be used to detect a file being read that does not conform to the current file format definition.

Checking the *HdrVersion* field before using a MAKE

utility also provides a convenient way to guard against the effects of "blind" recompilation. For example, suppose the file format has been extended to include new fields. Simply running MAKE against the new header file can produce a program that generates files with the new size data structures and corresponding version numbers, but nevertheless is invalid because of initialization problems.

The problem is that the newly-defined fields must be properly initialized, which is very unlikely to occur with code that was written before these fields were defined. For example, the proper way to default the AvLCim.DCFId field is to -1. However, there is no method for software that predates this field to provide the correct default.

Type And SubType Fields

xe "Version control strategy:Type and Subtype fields F-"§

The *AvLStrm* data structure contains two fields *AvLStrm.Type* and *AvLStrm.SubType*, whose purpose is to describe the kind of data that is contained in a stream. These fields can also be used to make future extensions to the file format. These fields have a limited set of defined values. An unknown *AvLStrm.Type* or *AvLStrm.SubType* value indicates that the file format has been extended to allow the presence of data whose interpretation is totally unknown. Existing software, encountering such a value, should ignore the stream's data.

File Organization

The file structure organization illustrated in Figure F-1 consists of: standard file header, *AvLFile* header, stream headers (one per stream), substream headers (minimum one per substream), frame data, and a frame directory.

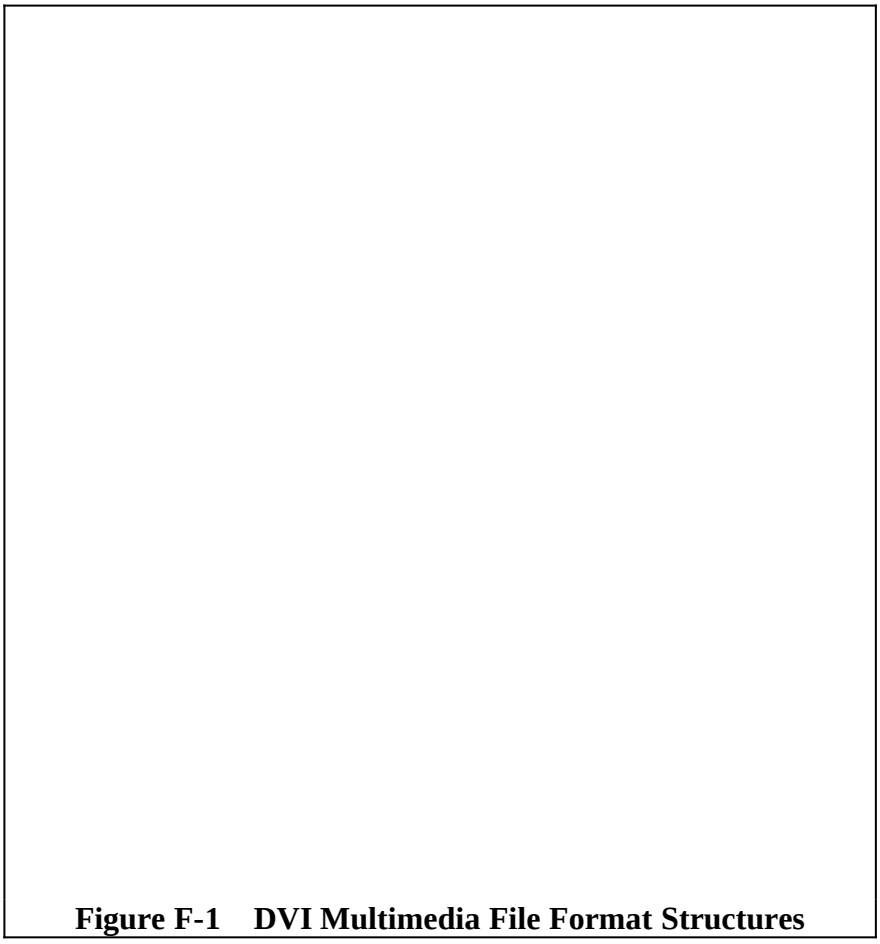


Figure F-1 DVI Multimedia File Format Structures

Standard File Header

xe "Standard file header F-"§

The first two entries in any file consist of a standard file header and an *AvLFile* data structure.

xe "Data structure:Standard file header F-"§

```
typedef struct

{
  U32 FileId;
  I16 HdrSize, HdrVersion;
  U32 AnnOffset;
} StdFileHdr;
```

The fields in the *StdFileHdr* data structure are:

Type	Field Name	Setting/Comments
U32	FileId	Must be set to VSTD_HDR_ID, which equals 0x56445649 (that is, VDVI) and should be validated.
I16	HdrSize	Should be set to sizeof(*StdFileHdr*), which is 12. On playback, this field should be used as described in the section, "Version Control Strategy". Since files do exist that have this field incorrectly set, it is recommended that files with *StdFileHdr.HdrVersion* = 1 ignore this field, and respond as if this field is set to 12.

I16	HdrVersion	Must be set to VSTD_HDR_VER, which is 1. On playback, this field value should be validated.
U32	AnnOffset	Can be set to zero when creating a file, and can be ignored on playback. It can also be set to point to an otherwise unused portion of the file, and unstructured data placed there. This pointer could be useful for adding copyright notices to the file.

AvLFile Header

xe "AvLFile header F-"§

The *AvLFile* data structure always follows immediately after the *StdFileHdr*.

xe "Data structure:File header F-"§

```
typedef struct
{
  U32 HdrID;
  I16 HdrSize, HdrVer, StrmGrpCnt, StrmGrpSize;
  U32 StrmGrpOffset;
  I16 StrmGrpVer, StrmSize, StrmVer, StrmCnt;
  U32 StrmOffset, HdrPoolOffset;
  I32 LabelCnt;
  U32 LabelOffset;
  I16 LabelSize, LabelVer;
  U32 VshOffset;
  U16 VshSize;
  I16 FrmVer;
  I32 FrmCnt, FrmSize;
  U32 FirstFrmOffset, EndOfFrmsOffset;
  I16 FrmHdrSize, FrmDirSize;
  U32 FrmDirOffset;
  I16 FrmDirVer, FrmsPerSec;
```

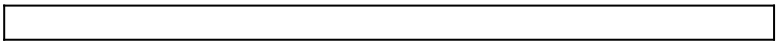
```
  U32 Flag;
  U32 FreeBlockOffset;
  U8 Patch[32];
} AvLFile;
```

The *AvLFile* header is the master directory of data structures within the file.

The fields in the *AvLFile* header data structure are:

Type	Field Name	Setting/Comments
U32	HdrID	Must be set to AVL_FILE_ID, which is 0x41565353 (that is, "AVSS"), and should be validated on playback.
I16	HdrSize	Should be set to sizeof(*AvLFile*), which is 120. This field should be used on playback as described in the section, "Version Control Strategy".
I16	HdrVer	Should be set to AVL_FILE_VER, which is 3. On playback, the value in the file must be less than or equal to this value.
I16	StrmGrpCnt	Should be set to zero, and need not be validated on playback.
I16	StrmGrpSize	Should be set to sizeof(*AvLStrmGrp*), which is 28, and need not be validated on playback. The *AvLStrmGrp* data structure is not described in this appendix.
U32	StrmGrpOffset	Should be set to zero, and need not be validated on playback.
I16	StrmGrpVer	Must be set to AVL_STRMGRP_VER, which is 3, and need not be validated on playback.
I16	StrmSize	Must be set to sizeof(*AvLStrm*), which is 44. This field should be used on playback as described in the section, "Version Control Strategy" for the *HdrSize* field.
I16	StrmVer	Should be set to AVL_STRM_VER, which is 3, and should be used on playback as described in the section, "Version Control Strategy" for the *HdrVersion* field.

Type	Field Name	Setting/Comments
I16	StrmCnt	The number of streams in the file. This value can be used as the *StreamCount* parameter with **AvkGrpBufCreate**. A stream typically consists of a set of bytes that describes a sequence of images or waveform audio samples. Each stream type is described in a separate section later in this appendix.
U32	StrmOffset	The offset of the array of *AvLStrm* structures. Usually set *to sizeof(StdFileHdr) + sizeof(AvLFile)* that is, the offset that points immediately after the *AvLFile*. Other offset values could also be used, provided the array of *AvLStrm* headers is placed in the file so that this offset points to it.
U32	HdrPoolOffset	The offset of a pool of substream headers, described in the section, "Substream Headers". This field should be set to point to this pool when a file is created. Typically, this pool begins immediately after the array of *AvLStrm* headers for *StrmCnt*, so its value could be set to *AvLFile.StrmOffset + (AvLFile.StrmCnt * sizeof(AvLStrm))* This field need not be used during playback, since the first substream header for each stream can be located through *AvLStrm.FirstHdrOffset*.
I32	LabelCnt	Should be set to zero, and need not be validated on playback.
U32	LabelOffset	Should be set to zero, and need not be validated on playback.
I16	LabelSize	Should be set to sizeof(*AvLLabel*), which is 20, and need not be validated on playback.
I16	LabelVer	Should be set to AVL_LABEL_VER, which is 3, and need not be validated on playback.



Type	Field Name	Setting/Comments

U32	VshOffset	The offset of the video sequence header (VSH) for this file. If none of the streams in this file require a VSH, this field and *AvLFile.VshSize* are zero. The VSH contains data required for the decompression of all PLV video streams in the file. The VSH data is passed to AVK through the *pWorkData* parameter of **AvkVidStrmFormat**. When creating new files under AVK with RTV, set this field to zero. AVK applications can also create files by combining audio/video data from existing or newly created files. Such file editing is legitimate under AVK, provided that no stream uses more than one compression algorithm. However, there are practical difficulties associated with generating a valid VSH. The data in the VSH depends on the actual images, and might differ from file to file, even if the files were compressed with the same PLV compression algorithm. Moreover, the file format only allows for a single VSH per file. Therefore, the merger of one or more video streams into a single file requires: m Combining the original VSHs into a new VSH m Modification of the compressed bitstreams While the PLV algorithms contain sufficient information to implement such software, the process is complex. Alternatively, the DOS media preparation utility called **VAvEd** can be used to create a properly merged VSH from several input video streams. See the **Media**

		Preparation Utility Reference For DOS for details on **VAvEd**.
U16	VshSize	The length of the VSH stored in the file. It is passed to AVK through the *pWorkDataSize* parameter of **AvkVidStrmFormat**. When creating a file whose only video is an RTV 2.0 or RTV 2.1 stream, this field should be set to zero. If you create a file with a validly formed VSH, then its size should be stored here.

Type	Field Name	Setting/Comments
I16	FrmVer	Should be set to AVL_FRM_VER, which is 3. On playback, the value in the file must be less than or equal to this value.
I32	FrmCnt	Should be set to the number of *AvLFrm* headers in the file. Typically, this field is initialized after all the frame data has been written into the file. On playback, an application can use this field (or *AvLFile.EndOfFrmsOffset*) to determine when to stop delivering data to **AvkGrpBufWrite**.
I32	FrmSize	The size of a frame (frame header plus data for all streams). This field is set to zero if the frames in a file have variable length, as is typical of motion video and audio files. If, however, all frames have exactly the same length, this field contains that length. Such a file could be generated by using an optional parameter with a DOS media preparation utility called **VLayout** (see the **Media Preparation Utility Reference For DOS** for details on **VLayout**). Stream sizes per frame can vary, but the sum of all stream data per frame must be fixed in order for this field to be non-zero.

U32	FirstFrameOffset	The offset to the first frame of interleaved stream data. The interleaved data consists of a sequence of *AvLFrm* headers. For playback, this interleaved data should get passed to **AvkGrpBufWrite**, in order to play the file from the beginning. For capture, **AvkGrpBufRead** is used to extract the frame data from AVK, which is then formatted into *AvLFrm* headers for storage in the file. When creating a file, the frame data is placed towards the end of the file, and an appropriate offset stored in this field. On playback, this offset is used to locate the first frame data. To start playing the file from some other point, the appropriate first *AvLFrm* must be located. This location process can be done either by parsing through the *AvLFrm* headers, or by using data stored within the optional frame directory.

Type	Field Name	Setting/Comments
U32	EndOfFrmsOffset	Must be set to the offset to the first byte after the frame data. When creating a file, its value is typically entered after the last byte of frame data has been entered into the file. On playback, no data located at or after this address should ever be passed to **AvkGrpBufWrite**.
I16	FrmHdrSize	The size of the frame header used for all frames. This field must be set to the length of the frame header, which is a value computed as sizeof(AvLFrm) + 4 * (AvLFile.StrmCnt - 1) The "-1" is needed because the *AvLFrm* data structure, as defined, already accounts for the presence of one stream. This field does not have to be validated when a file is played back. A better check could be implemented using the *AvLFrm.ChkSum* field, described in the section, "Frame Header".
I16	FrmDirSize	Must be set to sizeof(*AvLFrmDir*), which is 4.
U32	FrmDirOffset	The offset to the frame directory. The frame directory provides information that allows random access to an arbitrary frame within the file. It is recommended that all new files contain a frame directory, since it is very useful for random access. Some older files will, however, contain a zero for this field, meaning that the frame directory is missing. A DOS media preparation utility called **VAvCopy** can be used to add a frame directory to such files (see the **Media Preparation Utility Reference For DOS** for details on **VAvCopy**). Typically, the frame directory is physically placed immediately after the frame data.

I16	FrmDirVer	Must be set to AVL_FRMDIR_VER, which is 3. On playback, the value in the file must be less than or equal to this value.

Type	Field Name	Setting/Comments
I16	FrmsPerSec	Must be set to the frame per second rate, rounded to the nearest integer written. By convention, a value of 25 means precisely 25 fps, while any other value is adjusted by the fact that NTSC is 29.97 frames per second, not 30 frames per second. If a file is based on PAL original material, but intentionally has a frame rate that is not 25 frames per second, then a pad stream must be created in order to specify a frame rate that does not have this NTSC adjustment. The use of this field during playback to help derive *FrameRates* to pass to AVK is described in the section, "Derived Values".
U32	Flag	Should be set to AVL_FILE_INP_UPDATE while a file is being created, and set to zero before the file is closed. If a file is read when this field is non-zero, the data in the file might be incomplete, and should not be used.
U32	FreeBlockOffset	Should be set to zero when creating a file, and need not be validated during file playback.
U8	Patch[32]	Should be set to all zeroes when creating a file, and need not be validated during file playback.

Stream Header

xe "Stream header F-"§

AvLFile.StrmOffset holds the offset to an array of *AvLStrm* data structures, one for each of the *AvLFile.StrmCnt* streams in the file. The position in the array defines the stream number.

The *AvLStrm* data structure is:

xe "Data structure:Stream header F-"§

```
typedef struct
{
  U32 HdrID;
  U16 Type, SubType;
  I16 HdrCnt, NextStrmNum, StrmGrpNum, Pad;
  U32 Flag;
  I32 FrmSize;
  U32 FirstHdrOffset;
  U8 StrmName[16];
} AvLStrm;
```

This data structure describes the general nature of the data in a single stream, and points to more detailed information the substream header.

The fields in the *AvLStrm* data structure are:

Type	Field Name	Setting/Comments
U32	HdrID	Must be set to AVL_STRM_ID, which is 0x5354524d (that is, "STRM"), and need not be validated on playback.

U16 Type — Identifies the type of stream data. The stream type contains the following values:

Type	Value	Description
AVL_T_AUD	2	Compressed audio stream
AVL_T_CIM	3	Compressed image stream
AVL_T_ULAY	5	Associated per-frame data
AVL_T_UIM	6	Uncompressed Image Stream
AVL_T_PAD	7	Pad Stream

If a stream has some other value for *AvLStrm.Type*, its data can and should be ignored.

Type	Field Name	Setting/Comments
U16	SubType	These values depend on the value of *AvLStrm.Type*, and are described in the various sections on substream headers (*AvLCim*, *AvLULay*, *AvLUim*, and *AvLPad*).
I16	HdrCnt	Specifies the number of substream headers associated with this stream. This field should be set to one.
I16	NextStrmNum	Should be set to AVL_STRMGRP_END, which is -1, and need not be validated on playback.
I16	StrmGrpNum	Should be set to zero, and need not be validated on playback.
I16	Pad	Should be set to zero, and need not be validated on playback.
U32	Flag	Should be set to 0x4, if the value in *AvLStrm.FrmSize* is variable. This field need not be validated on playback.
I32	FrmSize	The maximum amount of data per frame in a stream. This field does not include the frame header size. For example, for the X stream, it is the size of the frame, and for the Y stream, it is the size of the largest Y data component for that stream. When creating a file, this value could be computed while the frame data is stored into the file, and then updated into the header after all the frame data has been written. This field is useful in estimating the maximum size of a frame, as described further in the section, "Derived Values".

Type	Field Name	Setting/Comments
U32	FirstHdrOffset	The offset to the stream header for this stream. The data structure at this offset must correspond to the *AvLStrm.Type* value. This data structure contains additional information about the stream, and will be described further in the section, "Substream Headers".
U8	StrmName[16]	A null-terminated ASCII string for the stream name. *StrmName* is not used by AVK playback, and can be set to all zeroes (which is interpreted as a null string). It is helpful, however, to set this field for use with the output of a DOS media preparation utility called **VAvCheck**. See the **Media** Preparation Utility Reference For **DOS** for details on **VAvCheck**.

Substream Headers

xe "Substream headers F-"§

An *AvLStrm* data structure contains general information about a stream. Type-dependent information is stored in substream headers of the following type: *AvLAud*, *AvLCim*, *AvLUlay*, *AvLUim* and *AvLPad*.

All the substream headers are located in a pool pointed to by *AvLFile.HdrPoolOffset*. The pool is located near the beginning of the file, to minimize the amount of seeking while the frame data is being processed. This is especially useful for files stored on devices like a CD-ROM that have comparatively slow seek times.

AvLAud: The Audio Substream Header

xe "Audio substream header F-"§

The *AvLAud* substream header describes the global characteristics of an audio stream.

xe "Data structure:Audio substream header F-"§

```
typedef struct
{
  U32 HdrID;
  I16 HdrSize, HdrVer;
  U8 OrigFile[80];
  I32 OrigFrm;
  I16 OrigStrm, Pad;
  I32 FrmCnt;
  U32 NextHdrOffset;
  U8 Lib[16], Alg[16];
  I32 Parm1;
  I16 Parm2, Parm3, LeftVol, RightVol;
  I32 LoopOffset, StartFrm;
  U32 Flag;
  I16 Parm4, Pad2;
  I32 DCFId;
} AvLAud;
```

For an audio stream, *AvLStrm.SubType* should be set to and validated for the value zero.

The fields in the *AvLAud* data structure are:

Type	Field Name	Setting/Comments
U32	HdrID	Should be set to AVL_AUD_ID, which is 0x41554449 (that is, "AUDI"), and should be validated on playback.
I16	HdrSize	Should be set to sizeof(*AvLAud*), which is 168. This field should be used on playback as described in the section on version control strategy.
I16	HdrVer	Should be set to AVL_AUD_VER, which is 5. On playback, the value in the file must be less than or equal to this value.
U8	OrigFile[80]	Should be set to all zeroes, and need not be validated on playback.
I32	OrigFrm	Should be set to zero, and need not be validated.
I16	OrigStrm	Should be set to zero, and need not be validated.
I16	Pad	Should be set to zero, and need not be validated on playback.
I32	FrmCnt	The number of frames.
U32	NextHdrOffset	The offset to the next substream header for this stream. This field should be set to AVL_LAST_HDR, which is 0x7ffffff, or zero for the last header.
U8	Lib[16]	Should be set to all zeroes, and need not be validated.

U8	Alg[16]	When creating a file, this field should be set to a null-terminated text string that identifies the audio compression algorithm. On playback, this field is used to derive other quantities that are passed to AVK to control playback, as described in the section, "Derived Values".

Type	Field Name	Setting/Comments
I32	Parm1	Should be set to the audio data rate in bits per second. This value is related to the *SamplesPerSecond* value used with **AvkAudStrmFormat**, as described in the section, "Derived Values".
I16	Parm2	The filter cutoff frequency to be used with the audio. This should be set to zero.
I16	Parm3	Should be set to zero.
I16	LeftVol RightVol	Should be set to 100 on file creation. These fields are intended to enable an editor to modify the volume level associated with an audio stream. On playback, these volume level numbers should be treated as a percentage of full volume, and used to form a multiplier with the application-specified volume before that volume is passed to **AvkAudStrmVolume**. In a few old files, this value was set to 4096. If 4096 is found in a file, it should be treated as if it were 100.
I32	LoopOffset	Should be set to -1, and need not be validated on playback.
I32	StartFrm	Should be set to zero, and need not be validated on playback.
U32	Flag	Used to signify monophonic or stereo. This field is zero for mono and AVL_AUD_STEREO, which is 0x00004000, for stereo. Files might exist in which the 0x00008000 bit is set. This bit denotes an old format for adpcm4e stereo which cannot be played by AVK.

I16	Parm4	Should be set to the *FrameRate* used with **AvkAudStrmFormat** when audio compression was requested. Typically, this *FrameRate* is the same for all streams of the file, and so this field can be set to zero. For playback, the proper way to determine the audio FrameRate is described in the section, "Derived Values".
I16	Pad2	Should be set to zero, and need not be validated on playback.

Type	Field Name	Setting/Comments
I32	DCFId	Should be set to -1, and need not be validated on playback. This value denotes generation on an end-user platform. A value of zero means that the Digital Compression Facility (DCF) it was generated on is unknown. A current list of DCFId's can be obtained from compression services.

AvLCim: The Compressed Image And Compressed Video Substream Header

xe "Compressed image and compressed video header F-"§

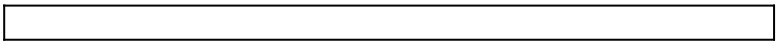
The *AvLCim* substream header is used for compressed motion video streams and compressed still images.

The compressed still images are distinguished by the use of:

- Specific values of *AvLStrm.SubType*
- Different values of *DeCodeAlg*

The various SubTypes that can be used are:

SubType	Value	Description
AVL_ST_Y	1	Y-channel image data
AVL_ST_U	11	U-channel image data
AVL_ST_V	12	V-channel image data
AVL_ST_YVU	13	YVU image data
AVL_ST_YUV_S	14	YUV image data (Industry Standard Order)



For the PLV algorithms, three streams (Y, V, U) are used to convey the information within one "logical" video stream, which explains the stream counting rules in **AvkVidStrmCreate** and **AvkAudStrmCreate**.

Two SubTypes, AVL_ST_YVU and AVL_ST_YUV_S, are used to hold sequences of images, in which the entire image is contained in a single stream. (The suffix "_S" in AVL_ST_YUV_S is used only to distinguish the difference between two otherwise very similar SubType names.) These SubTypes differ only in respect to the order in which the color components are stored.

All DVI video images use a YVU order for the color components, except for JPEG.

AVK does not support the playing of motion video for arbitrary streams of compressed images, but only for a few explicitly identified algorithms.

Such images can, however, be displayed by loading them into image buffers, using **AvkImgDecompress**, and using a suitable connector.

xe "Data structure:Compressed image and compressed video header F-"§

```
typedef  struct
{
  U32 HdrID;
  I16 HdrSize, HdrVer;
  U8 OrigFile[80];
  I32 OrigFrm;
  I16 OrigStrm, Pad;
  I32 FrmCnt;
  U32 NextHdrOffset;
  I16 XPos, YPos, XLen, YLen;
  I16 XCrop, YCrop, DropFrm, DropPhase;
  I32 StillPeriod;
  I16 BufsMin, BufsMax, DeCodeAlg, Pad2;
  I32 DCFId;
} AvLCim;
```

This substream data structure can be used to store several different kinds of compressed images. The fields in the *AvLCim* data structure are:

Type	Field Name	Setting/Comments
U32	HdrID	Should be set to AVL_CIM_ID, which is 0x43494d47 (that is, "CIMG"), and should be validated on playback.
I16	HdrSize	Should be set to sizeof(*AvLCim*), which is 136. This field should be used on playback as described in the section, "Version Control Strategy".

I16	HdrVer	Should be set to AVL_CIM_VER, which is 4. On playback, the value in the file must be less than or equal to this value.
U8	OrigFile[80]	Should be set to zero, and need not be validated at playback.

Type	Field Name	Setting/Comments
I32	OrigFrm	Should be set to zero, and need not be validated at playback.
I16	OrigStrm	Should be set to zero, and need not be validated at playback.
I16	Pad	Should be set to zero, and need not be validated at playback.
I32	FrmCnt	The number of frames until the next substream header applies. This field should be AVL_LAST_HDR, which is 0x7fffffff.
U32	NextHdrOffset	The offset to the next substream header for this stream. This field should be set to zero.
I16 YPos	XPos	Should be set to zero, and need not be validated on playback.
I16 YLen	XLen	Specifies the maximum width and height for the decompressed images in this file. An AVK application should set these fields properly, and validate that these fields will not overflow the limits imposed by the *Xres* and *Yres* passed to the **AvkVidStrmFormat** call. When using **AvkImgDecompress**, AVK should report an error if the destination image is too small. However, when playing motion video, the microcode might not detect an attempt to decompress an image into a bitmap too small to hold it, causing unpredictable results.
I16 YCrop	XCrop	Should be set to zero, and need not be validated on playback.
I16 DropPhase	DropFrm	Should be set to zero, and need not be validated on playback.

Type	Field Name	Setting/Comments
I32	StillPeriod	Indicates that the video was compressed such that every Nth frame was intraframe encoded. For example, if every image of the stream is a still image this field should be one. If this value is three, then frames numbered "0, 3, 6, 9, 12, ..." are all still images. If this value is one, then every image is a still frame. In addition to these intraframe coded images, the stream might contain additional intraframe encoded images. The default value of AVL_CIM_RANDOM_STILL, which is -1, indicates that intraframe image spacing is unspecified.
I16	BufsMin	Should be set to zero.
BufsMax		
I16	DecodeAlg	Should be set to the decompression algorithm. On playback, the *AvLCim.DecodeAlg* value should be passed to **AvkVidStrmFormat** or **AvkImgDecompress**.
I16	Pad2	Should be set to zero, and need not be validated on playback.
I32	DCFId	Operates the same as *AvlAud.DCFId*. This field should be set to -1, and need not be validated on playback.

AvLUlay: The Underlay Substream Header

xe "Underlay substream header F-"§

Underlay streams hold digital data associated with the same interval of time as the other streams that are present in each frame. Generally, each SubType can

have its own underlay substream header definition.

However, for many kinds of underlay data, the following generic underlay substream header can be used.

xe "Data structure:Underlay substream header F-"§

```
typedef  struct
{
  U32 HdrID;
  I16 HdrSize, HdrVer;
  U8 OrigFile[80];
  I32 OrigFrm;
  I16 OrigStrm, Pad;
  I32 FrmCnt;
  U32 NextHdrOffset;
  I32 DCFId;
} AvLUlay;
```

Only one underlay SubType is supported, to be used for holding SMPTE timecodes (see the section, "SMPTE Timecode Underlay Streams" for details). In addition, a range from zero through 32767 has been reserved for possible registration of specific, to-be-determined, well-defined uses. SubTypes greater than 32767 will not be controlled and can be freely used for application-specific purposes.

In a multiple stream file, several underlay streams can exist with the same SubType, each associated with another interleaved stream. In this case, by convention, the data in the underlay stream applies to the closest preceding stream of an appropriate type.

The fields in the *AvLULay* data structure are:

Type	Field Name	Setting/Comments
U32	HdrID	Should be set to AVL_ULAY_ID, which is 0x554e4452 (that is, "UNDR"), and need not be validated on playback.
I16	HdrSize	Should be set to sizeof(*AvLULay*), which is 108. This field should be used on playback, as described in the section, "Version Control Strategy".
I16	HdrVer	Should be set to AVL_ULAY_VER, which is 4. On playback, the value in the file must be less than or equal to this value.
U8	OrigFile[80]	Should be set to zero, and need not be validated at playback.
I32	OrigFrm	Should be set to zero, and need not be validated at playback.
I16	OrigStrm	Should be set to zero, and need not be validated at playback.
I16	Pad	Should be set to zero, and need not be validated at playback.
I32	FrmCnt	The number of frames until the next substream header applies. This field should be AVL_LAST_HDR, which is 0x7fffffff.
U32	NextHdrOffset	The offset to the next substream header for this stream. This field should be set to zero.
I32	DCFId	Operates the same as *AvlAud.DCFId*. This field should be set to -1, and need not be validated on playback.

AvLUim: The Uncompressed Image Substream Header

xe "Uncompressed image substream header F-"§

The *AvLUim* data structure is used to hold uncompressed images.

xe "Data structure:Uncompressed image substream header F-"§

```
typedef  struct
{
  U32 HdrID;
  I16 HdrSize, HdrVer;
  U8 OrigFile[80];
  I32 OrigFrm;
  I16 OrigStrm, Pad;
  I32 FrmCnt;
  U32 NextHdrOffset;
  I16 XPos, YPos, XLen, YLen, PixBits, Pad2;
  I32 DCFId;
} AvLUim;
```

The fields in the *AvLUim* data structure are:

Type	Field Name	Setting/Comments
U32	HdrID	Should be set to AVL_UIM_ID, which is 0x55494d47 (that is, "UIMG"), and need not be validated on playback.
I16	HdrSize	Should be set to sizeof(*AvLUim*), which is 124. This field should be used on playback, as described in the section, "Version Control Strategy".

I16	HdrVer	Should be set to AVL_UIM_VER, which is 4. On playback, the value in the file must be less than or equal to this value.
U8	OrigFile	Should be set to zero, and need not be validated at playback.
I32	OrigFrm	Should be set to zero, and need not be validated at playback.

Type	Field Name	Setting/Comments
I16	OrigStrm	Should be set to zero, and need not be validated at playback.
I16	Pad	Should be set to zero, and need not be validated at playback.
I32	FrmCnt	The number of frames until the next substream header applies. This field should be AVL_LAST_HDR, which is 0x7fffffff.
U32	NextHdrOffset	The offset to the next substream header for this stream. This field should be set to zero.
I16 YPos	XPos	Should be set to zero, and need not be validated on playback.
I16 YLen	XLen	Specifies the maximum width and height for the decompressed images in this file.
I16	PixBits	Should be set to the average number of bits per pixel in the image. Typical values for this field are 8, 9, 16 and 24. On playback, *AvLUim.PixBits* should be used to determine the value of *BitmapFormat* to pass on a call of **AvkImgCreate**.
I16	Pad2	Should be ignored and set to zero.
I32	DCFId	Operates the same as AvLAud.DCFId. This field should be set to -1, and need not be validated on playback.

AvLPad: The Pad Substream Header

xe "Pad substream header F-"§

Pad streams files are generated by a DOS media preparation utility called **VLayout**. See the **Media Preparation Utility Reference For DOS** for details on **VLayout**. A pad steam header can tore a frame rate more accurately than can be done by using *AvLFile.FrmsPerSec*.

VLayout generates pad data in the various *AvLFrm* headers, such that the average rate of data consumption precisely matches the standard data rate from a CD-ROM (153,600 bytes per second). For **VLayout**, this pad data is set to all zero.

On playback, most of the data in pad streams can be ignored on playback. However, any existing pad streams should be used to derive accurate *FrameRates* for all streams during playback, as described in the section, "Derived Values".

xe "Data structure:Pad substream header F-"§

```
typedef struct
{
  U32 HdrID;
  I16 HdrSize, HdrVer;
  U8 OrigFile[80];
  I16 OrigStrm, Pad;
  I32 FrmCnt;
  U32 NextHdrOffset;
  I32 ImagesPer, Seconds, VidFast, VidVar, VidRev, VidStart;
  I16 UlayFast, UlayVar, UlayRev, UlayStart;
  I16 PipeDepth, PipeStart, MinSeek, MinPad;
  I32 DCFId;
} AvLPad;
```

The fields of the *AvLPad* data structure are:

Type	Field Name	Setting/Comments
U32	HdrID	Should be set to AVL_PAD_ID, which is 0x50414421 (that is, "PAD!"), and should be validated on playback.

I16	HdrSize	Should be set to sizeof(*AvLPad*), which is 144. This field should be used on playback as described in the section, "Version Control Strategy".

Type	Field Name	Setting/Comments
I16	HdrVer	Should be set to AVL_PAD_VER, which is 4. On playback, the value in the file must be less than or equal to this value.
U8	OrigFile[80]	Should be set to zero, and need not be validated at playback.
I16	OrigStrm	Should be set to zero, and need not be validated at playback.
I16	Pad	Should be set to zero, and need not be validated at playback.
I32	FrmCnt	The number of frames until the next substream header applies. This field should be AVL_LAST_HDR, which is 0x7fffffff.
U32	NextHdrOffset	The offset to the next substream header for this stream. This field should be set to zero.
I32	ImagesPer Seconds	Two 32-bit integers whose ratio is the frame rate in images per second. These fields should be set to zero, and need not be validated on playback On playback, these fields are used to derive the frame rate for all streams in the file, as described in the section, "Derived Values".
I32	VidFast	Should be set to zero, and need not be validated on playback.
I32	VidVar	Should be set to zero, and need not be validated on playback.
I32	VidRev	Should be set to zero, and need not be validated on playback.
I32	VidStart	Should be set to zero, and need not be validated on playback.
I16	UlayFast	Should be set to zero, and need not be validated on playback.
I16	UlayVar	Should be set to zero, and need not be validated on playback.

I16	UlayRev	Should be set to zero, and need not be validated on playback.

Type	Field Name	Setting/Comments
I16	UlayStart	Should be set to zero, and need not be validated on playback.
I16	PipeDepth	Should be set to zero, and need not be validated on playback.
I16	PipeStart	Should be set to zero, and need not be validated on playback.
I16	MinSeek	Should be set to zero, and need not be validated on playback.
I16	MinPad	Should be set to zero, and need not be validated on playback.
I32	DCFId	Operates the same as AvLAud.DCFId. This field should be set to -1, and need not be validated on playback.

SMPTE Timecode Underlay Streams

xe "SMPTE timecode underlay streams F-"§

If the SubType of an underlay stream is AVL_ST_TIMECODE (which is 1), the stream contains SMPTE timecode data. The data in each frame consists of four bytes which are the Binary-Coded-Decimal representation of the HH:MM:SS:FF for that frame, as defined by the SMPTE standard for time codes.

Generally, several interleaved audio and/or video streams can exist in a file, each with its own timecode data. The rule for associating a timecode stream with audio or video data is that the timecode stream refers to the immediately preceding audio or video stream.

For example, a file compressed by compression services might have the following six stream types: Y, V, U, Timecode, Audio, Timecode. This stream order indicates that there is valid (and possibly equal) timecode information for both the video and audio data.

Typically, the data to fill this stream is extracted from a time code reader at the same time as the original video and audio are digitized.

Frame Data

Each frame of data in a DVI Multimedia file is preceded by a frame header, identifying the amount of data per stream.

Frame Header

xe "Frame header F-"§

This data structure is used to introduce the actual data of the file. All the other headers only describe this data. The data consists of a sequence of contiguous *AvLFrm* header/data pairs, one for each frame of the file. See Appendix C, "Algorithm Characteristics", for details on interpreting data during playback or capture.

xe "Data structure:Frame header F-"§

```
typedef struct
{
  I32 FrmNum, RevOffset, ChkSum;
//I32 StrmFrmSize[AvLFile.StrmCnt]  This is invalid C syntax.
  I32 StrmFrmSize[1];  //Note, This line has valid C syntax, but has wrong
array size
} AvLFrm;
```

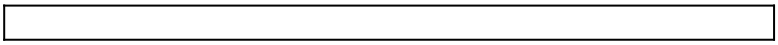
The fields of the *AvLFrm* data structure are:

Type	Field Name	Setting/Comments
I32	FrmNum	The sequential frame number in each file, starting with zero, allowing several files to be opened and fed in sequence to AVK. This field is not used by AVK playback, in order to allow convenient concatenation of the data from several files. This field is generated by AVK capture, for use in identifying the precise time that each frame's data occurred.
I32	RevOffset	The file offset to the previous *AvLFrm* in the file (measured from the beginning of the file). For the first frame of a file, this file offset is zero. This field must be properly generated on file creation. While *AvLFrm.RevOffset* is typically ignored on AVK playback, this offset is used by some DOS media preparation utilities provided with ActionMedia II software.

I32	ChkSum	Provides an efficient check of whether or not a given block of data begins with a valid *AvLFrm*. Its value is formed by exclusive ORing all *AvLFile.FrmHdrSize* 32 bit words in the frame header (excluding this one) with the constant AVL_FRM_ID, which is 0x46524d48 (that is, "FRMH"). This field must be computed by the application before storing the data received by **AvkGrpBufRead** into a file. This field can be validated by AVK when the data is passed to **AvkGrpBufWrite**.

Type	Field Name	Setting/Comments

I32	StrmFrmSize	Consists of one long word for each of the *AvLFile.StrmCnt* streams in the file. This field contains the byte count for the actual data within each stream of the file. This frame data immediately follows the *AvLFrm*, with no padding between the frame data for successive streams. By convention, there are some special *AvLFrm.StrmFrmSize* values, that can be used with audio streams and compressed image streams. The size values zero, eight and sixteen indicate that no data exists for this frame. These byte values have slightly different interpretations by AVK playback, namely missing, silent audio and transparent video frames, respectively. The value eight is only used for audio streams, and denotes one frame time's worth of silence. The associated eight bytes of actual data, may be set to zero. The value sixteen is used only for compressed image streams, and denotes one frame time in which the image on the display does not change. The associated 16 bytes of data must be a valid compressed image bitstream header. This header is described in the section, "Compressed Image Bitstream Header". In this context, the compressed image bitstream header consists of eight words with the values: *AvLCim.DecodeAlg*, 0, 128, 0, 0, 0, image_height_in_pixels, image_width_in_pixels. A byte count of zero provides physical spacing of the frame data, but assumes that some other mechanism will be used to control the temporal spacing.



Compressed Image BitStream Header

xe "Compressed image bitstream header F-"§

For compressed image streams, the actual data for each frame begins with a 16-byte bitstream header. For PLV sequences, the image data is contained in three separate streams that do not necessarily have the same values for these fields. This bitstream header must be present for all compressed images. If the compression is imported from another system, such as a JPEG image, then a suitable bitstream header must be synthesized and pre-pended. While in theory, a third party JPEG-conformant decompression processor should ignore this header, it is probably a good idea to strip this header before exporting JPEG images.

xe "Data Structure:Compressed image bitstream header F-"§

```
typedef struct
{
  U16 AlgNum;
  U16 Flags;
  U32 NumBits;
  U32 AlgSpec;
  U16 YSize;
  U16 XSize;
} AvLBsh;
```

The fields of the *AvLBsh* data structure are:

Type	Field Name	Setting/Comments
U16	AlgNum	Contains the AlgName (as passed to **AvkVidStrmFormat**) used to compress the image.

Type	Field Name	Setting/Comments

U16 Flags — Contains information that might be used by editor programs and for random access. This flag word contains various bits that might be useful for making decisions about individual frames.

All bits and bit combinations not explicitly described in the following questions are reserved for future use, and should be masked away before making any of the following decisions.

m Is this image intraframe encoded (that is, a still frame)?

The status of an image intraframe can always be determined by examining 0x4. If this bit is set, this image can be decompressed without reference to any other image. The first frame of a file should be a still frame.

m Can this image be used as the last image of a self-contained edited subsequence?

For simple compression algorithms, the answer to this question is always yes. However, in the class of algorithms not supported by AVK, the compressed data in the bitstream might not be used until a subsequent image has also been decompressed. To write code that will also work for such compression algorithms, editing programs should determine this image use as follows:

- The answer is yes if bit 0x80 is zero.
- The answer is also yes if bit 0x80 is one and bit 0x40 is one.

m Can this image be replaced by a transparent image, with no effect on any other images?

If bit 0x80 is zero, bits 0x300 contain a two bit count of the number of images until the next reference frame. That is, if the 0x300 bits are 01, this image might be discarded. If they are 10, this image and the next image might be discarded If they are 11, this image and the next two images might be discarded. If the 0x300 bits are 00, there is no information about the distance to the next intracoded image. In addition, if *AvLCim.StillPeriod* is not AVL_CIM_RANDOM_STILL, *AvLCim.StillPeriod* can be used to predict the distance to the next still image.

If bit 0x80 is one, an image can be discarded if the 0x700 bits are 000.

Type	Field Name	Setting/Comments
U32	NumBits	Contains the number of bits in the image, including this header. For historical reasons, RTV 1.0 and RTV 1.5 contain a byte count instead.
U32	AlgSpec	Contains information related to the use of the VSH data with this image. For compression algorithms that do not require a VSH for decompression, this field will be zero.
U16	YSize XSize	The height and width (respectively) of the image in pixels. For the subsampled U and V PLV streams, these fields describe the height and width of the subsampled chrominance bitmap.

Compressed Audio Bitstream Header

xe "Compressed audio bitstream header F-"§

The frame data for an audio bitstream has an internal structure. Knowledge of this structure is useful for conversion between audio bitstream formats, for editing files containing audio.

xe "Data Structure:Compressed audio bitstream header F-"§

```
typedef struct
{
  I16 Word1;
  I16 Word2;
  I16 Word3;
  I16 Word4;
}
```

This header is generated automatically by AVK when it digitizes and compresses. However, if a file is

edited, or a bitstream is converted from another source, the following information is needed to generate a valid frame.

The fields of the compressed audio bitstream header data structure are:

Type	Field Name	Setting/Comments
I16	Word1	The number of words in this frame, not counting the four words of this header.
I16	Word2, byte1	The audio algorithm, which is encoded as follows: adpcm4e-mono 1 adpcm4e-stereo 3 pcm8-mono 5 pcm8-stereo 7
	Word2, byte2	This field of a four-word header is always 0xFF. An audio stream that contains some other value implies that the audio bitstream did not have a four-byte header. Some adpcm4e files exist that do not have this header. The format of audio streams that are missing the audio bitstream header are not described in this appendix. AVK will automatically detect the absence of the audio bitstream header, and apply appropriate defaults in order to play those bitstreams.
I16	Word3	This field contains the sample rate in samples per second.
I16	Word4	This field should be set to zero.

The remaining audio data in a frame consists of a concatenated sequence of subframes. A monophonic subframe consists of precisely 32 bytes, while a stereo subframe consists of precisely 64 bytes. Frames contain an integral number of subframes. Ideally, for a given algorithm and sample rate, the number of subframes per frame would be a constant, generated by a simple formula (bits-per-second divided by bits-per-subframe).

In practice, this formula yields a non-integral value. To deal with this non-integral value, "occasionally" a frame will have an extra subframe so that audio bit rate will average to the right value. When converting audio data from another file format to play under AVK, these extra subframes must be inserted, because AVK uses this average bit rate to maintain lip-synch.

An audio stream can be edited by splicing together subframes. After editing:

- A valid 4-word audio bitstream header must be generated at the beginning of each frame.
- The number of subframes per frame must be adjusted so that the average bit rate is correct.
- In addition, when splicing adpcm4e subframes, the first 16 bits of data for the first subframe of a "cut" must be zero. A "cut" is the beginning of a fragment of continuous audio.

The data in the subframes depends on the audio algorithm. The adpcm4e bitstream format inside a subframe is not documented in this appendix, but is available. The pcm8 bitstream format is very simple. A mono bitstream consists of a sequence of 8-bit values representing the instantaneous volume level. For a stereo bitstream, the data consists of a sequence of two-byte values, each of which represents the instantaneous volume to the left and right speakers

respectively.

Frame Directory

xe "Frame directory F-"§

The frame directory is useful for random access to a file, and consists of one *AvLFrmDir* header for each *AvLFrm* header of the file. This data is typically (but not necessarily) written towards the end of the file, after all the *AvLFrm* header data has been entered.

Since this data structure has a fixed length, and is present for every frame, it is possible to compute the location of a specific *AvLFrmDir* header in the file, and seek directly to it.

xe "Data structure:Frame directory F-"§

```
typedef struct
{
  U32 FrmOffset;
} AvLFrmDir;
```

The 31 least significant bits of *AvLFrmDir.FrmOffset* is the offset to the associated *AvLFrm* header in the file.

The high order bit is 1, if this frame can be used for a random access to the frame data of every stream in the file. The first frame of a file must be usable for random access.

This bit can be set by using information about each compressed image, as described in the section, "Compressed Image Bitstream Header".

Theoretically, an audio stream is only suitable for random access if it has the same *FrameRate* as the video streams with which it is interleaved. However, in practice, all the audio algorithms supported will quickly resynchronize, even if started in the middle of an audio frame.

A frame directory can also be generated through use of an optional parameter to the DOS media preparation utility **VAvCopy**. See the **Media Preparation Utility Reference For DOS** for details on **VAvCopy**.

Derived Values

xe "Derived values F-"§

The preceding sections describe the values explicitly encoded into the current file format. This section summarizes the rules for deriving the values of several quantities not explicitly present in a file.

AVK Frame Rate

xe "AVK frame rate F-"§

xe "Frame rate F-"§

In AVK, the frame rate received is designated as microseconds per frames. However, since the DVI multimedia file format designates frame rate as frames per second, an application must translate the designation.

When audio is combined with video in a file with interleaved streams, each chunk of audio and video data should have the same playing time. Therefore, explicitly chunking the audio data so that it has the same frame rate as the video allows cutting and pasting of interleaved audio/video data to form edited files. A DOS media preparation utility such as **VAvEd** can be used for this cutting and pasting process. See the **Media Preparation Utility Reference For DOS** for details on **VAvEd**.

For AVK, the frame rate of the file is also used to control the synchronization of audio and video data. To synchronize its play against a wall clock, each stream processed uses data that is structured into frames with a known playback duration.

PAL original material is also fully supported in AVK. Therefore, the following rules apply for determining the frame rate of the streams of an existing file:

- First, determine if the file has a pad stream. If a pad stream exists, the pad stream should be used to compute the frame rates of all streams in the file.

  To determine whether or not a file has a pad stream, examine the *AvLStrm* data structures, looking for a structure whose *AvLStrm.Type* is AVL_T_PAD (that is, 7).

- In rare cases, a file can contain several pad streams, in which case the last pad stream should be used for frame rate calculations. Since the *AvLStrm* header data is stored as an array of fixed length structures, it is straight forward to search all *AvLStrm* header data in reverse order.
- After identifying the pad stream, access *AvLPad.ImagesPer* and *AvLPad.Seconds*. These values can be converted to a frame rate in AVK units (microseconds per frame), using the formula:

```
FrameRate = Round ((1,000,000 *
        AvLPad.Seconds)/AvLPad.ImagesPer))
```

- If the file does not contain a pad stream, the frame rate of video steams is found by examining *AvLFile.FrmsPerSec*. This value is stored as an integer.

  To convert it to a frame rate, use the formula:

```
FrameRate = Round ((1,000,000/AvLFile.FrmsPerSec) *
        NTSC_ADJUSTMENT)
```

  The symbol NTSC_ADJUSTMENT is 1001/1000 or 1. The value "1" is only used when *AvLFile.FrmsPerSec* is exactly 25.
- If the file does not contain a pad stream, the frame rate for an audio stream can be determined by examining *AvLAud.Parm4*. If

AvLAud.Parm4 is not present (as indicated by the value in *AvLAud.HdrSize* or *AvLAud.HdrVer*), or if its value is zero, the audio frame rate is derived from *AvLFile.FrmsPerSec*, as described.

If *AvLAud.Parm4* is nonzero, the audio frame rate is derived by substituting *AvLAud.Parm4* for *AvLFile.FrmsPerSec* in the formula above.

When a file is created, all its streams should have the same frame rate. If this frame rate can be accurately represented via *AvLFile.FrmsPerSec*, there is no need to include a pad stream. However, if the frame rate cannot be regenerated by the above calculation, the file should be created with a pad stream so that an accurate frame rate can be stored within it.

Maximum Frame Size And Group Buffer Size

xe "Maximum frame size and group buffer size F-"§

When creating a group buffer for AVK, it is important to know the largest size of a frame that might be found in the file. While this information is not explicitly recorded in a file, a reasonable upper limit can be estimated by using the *AvLStrm.FrmSize* values in each *AvLStrm*. This upper bound is the sum of the *AvLStrm.FrmSize* fields from every stream of the file, plus the length of a frame header (that is, *AvLFile.FrmHdrSize*). Generally, *AvLFile.FrmHdrSize* is larger than sizeof(*AvLFrm*).

This calculation only yields an upper bound because it is possible that different streams achieve their

maximums at different positions in the file.

Pixel Aspect Ratio

xe "Pixel aspect ratio F-"§

All images, compressed or uncompressed, that can be stored within a file are assumed to have a 5:4 pixel aspect ratio. This pixel aspect ratio results from DVI technology's use of 256 x 240 images to store full screen images, and the fact that TV screens have a 4:3 width to height ratio.

The pixel aspect ratio should be considered when scaling an image to the display. This is no problem for AVK, since AVK only supports bitmaps with 5:4 pixel aspect ratio.

Audio Algorithm Number And Bits Per Sample

xe "Audio algorithm number F-"§

xe "Bits per sample F-"§

AVK requires the use of a 16-bit integer that identifies the audio algorithm, while the file contains a string naming the algorithm. The following table provides the correspondence between audio algorithm names and numbers:

AvLAud.Alg	AlgName Symbol	AlgName Value	BitsPerSample
adpcm4e	AVK_ADPCM	0x400	4
pcm8	AVK_PCM8	0x801	8

This table also indicates the number of bits per sample associated with each supported audio algorithm.

Audio SamplesPerSecond And AvLAud.Parm1

xe "Audio SamplesPerSecond F-"§

To play a file under AVK, **AvkAudStrmFormat** must be called with the audio data rate in units of samples per second, even though the file contains the data rate in bits per second. *SamplesPerSecond* can be calculated from the bits per second value in *AvLAud.Parm1* by dividing by the *BitsPerSample* value from the audio algorithm table. *Parm1* has the same value for mono and stereo. The only difference between mono and stereo is the stereo bit in *AvLAud.Flag*.

When capturing a file this calculation process is reversed. The value of *SamplesPerSecond* passed to **AvkAudStrmFormat**, and the bits per sample, are used to compute the proper bits per second value to store in the file as *AvLAud.Parm1*. The *AvLAud.Flag* stereo bit is set to indicate mono or stereo.

Still Image Formats

xe "Still image formats F-"§

This section describes the old file format for still images.

Still Image File Structure

xe "Still image files"§

Image files by convention use the following file extensions: IMY, IMV, IMU, CMY, CMV, CMU, I16, and C16.

In addition, the file header in the file describes whether or not the file contains compressed data and also the pixel format of the file's image data.

For 9-bit images, there are three separate files, each containing one component of the data.

That is, a filename such as abcdefgh.imy contains only the luminance component of the data, stored as an 8 bit per pixel bitmap.

The associated chrominance data would be stored in the separate files abcdefgh.imu and abcdefgh.imv,

each as eight bit per pixel images.

The file suffix corresponds to the color component in the file.

To process the data in these image files, an application would first read in the image file header, which is defined as follows:

xe "Data structure:Still image files"§

```
typedef struct
{
  U32 ImIDCode;
  I16 ImByteSize, ImVer;
  U32 ImAnnOffset;
  U32 ImPlaneFlag;
  U16 ImXLen, ImYLen, ImPixBits, ImCodeVer;
  U32 ImImageOff;
  U16 ImClutCnt, ImClutBits;
  U32 ImClutOff;
  U32 ImAppDataOff, ImAppDataSize;
  U32 ImImageSize;
  U16 ImColor, ImPlane;
  U32 pad2, pad3;
} AvLImHdr;
```

The first four fields are for the standard file header *StdFileHdr*, with modified field name, as described previously for files, and using the same version control strategies. However, unlike the standard file header, these fields are imbedded within the *AvLImHdr* and are used to control the versions of the entire header.

The fields of the image file header *AvLImHdr* data structure are:

Type	Field Name	Setting/Comments
U32	ImIDCode	Should be validated to be AVK_IM_FILE_ID, which is 0x56494d20 (that is, "VIM ").
I16	ImByteSize	Must be set to sizeof(*AvLImHdr*).
I16	ImVer	Should be validated to be AVK_IM_HDR_VER which is 5.
U32	ImAnnOffset	Can be set to zero when creating a file, and can be ignored on playback. It can also be set to point to an otherwise unused portion of the file, and unstructured data placed there. This pointer could be useful for adding copyright notices to the file.
U32	ImPlaneFlag	Not used.
U16	ImXLen ImYLen	The width and height of the image, measured in pixels. These values must be used with **AvkImgCreate**. For the chrominance planes of 9 bit images, these fields will indicate the 4:1 subsampling of the chrominance. This should be validated, since AVK does not support arbitrary subsampling ratios. The values to pass to **AvkImgCreate** should be the value for the luminance component. 24 bit images are also stored as three separate files, except that *AvLImgHdr.ImXLen* and *AvLImgHdr.ImYLen* do not indicate any subsampling for U and V.

U16	ImPixBits	Indicates this plane's pixel width in bits. This has the value 8 or 16. For a 9-bit image, there are three image component files, each of which has the value 8 for this field. This field must be used to derive the *BitmapFormat* parameter for **AvkImgCreate**.

Type	Field Name	Setting/Comments
U16	ImCodeVer	The algorithm number associated with this bitstream, provided it contains compressed data. If the file contains an uncompressed image, this field will be zero. The algorithm numbers used by the DVI multimedia file format to support still image algorithms are described in Appendix C, "Algorithm Characteristics".
U32	ImImageOff	The offset within the file to where the image data is placed. This offset should be used to access the image data. The data for an image is stored by row (that is, *AvLImHdr.YLen* pixels exist for the first line of the image, followed immediately (since no padding exists) by the next line of image data. For a 9 or 24 bit image, the data for each plane should be passed to **AvkImgWrite** or **AvkImgBufWrite** in the order Y, V, U, with no padding between planes.
U32	ImageSize	Specifies the amount of image data in the file, measured in bytes. This byte measurement is especially useful if the image data is compressed. For an uncompressed image, this field will be the same as the value calculated, using the specified height, width, and bits per pixel.
U16	ImClutCnt ImClutBits	These fields can be ignored as they are not supported under AVK.
U32	ImClutOff	This fields is not supported under AVK. However, if *ImClutOff* is not equal to zero, the file cannot be processed.
U32	ImAppDataOff ImAppDataSize	These fields can be ignored as they are not supported under AVK.

U32	ImImageSize	This field can be ignored as it is not supported under AVK.
U16	ImColor ImPlane	These fields can be ignored as they are not supported under AVK.
U32	pad2 pad3	These fields can be ignored as they are not supported under AVK.

Storing Still Images In The DVI Multimedia File Format

xe "Storing still images in DVI multimedia file format F-"§

The following still image file format descriptions provide a reference for using the DVI multimedia file format to store images that originated in various types of image files.

IMY, IMV, IMU Image Files

xe "Still image file format storage :IMY, IMV, IMU image files F-"§

xe "Storage:Still image files F-"§

The data from these image files is combined into a single stream, with the following stream and substream header values:

Type	AVL_T_UIM
Sub-Type	AVL_ST_YVU
PixBits	9 or 24

The uncompressed image data from the three files is concatenated one after the other, in the order Y, V, U, with no intervening padding.

CMY, CMV, CMU Image Files

The data from these files is combined into a single stream with the following stream and substream header values:

Type	AVL_T_CIM
Sub-Type	AVL_ST_YVU
DeCodeAlg	128 or 129 (as read from AvkImHdr.ImCodeVer)
StillPeriod	1

The compressed image data from the three files would be concatenated one after the other, in the order Y, V, U, with no intervening padding.

The *BitmapFormat* used to store these images can be inferred from *DecodeAlg*, since all currently supported decompression algorithms are associated with a single *BitmapFormat*.

I16 Image Files

Type	AVL_T_UIM
Sub-Type	AVL_ST_YVU
PixBits	16

C16 Image Files

Type	AVL_T_CIM
Sub-Type	AVL_ST_YVU
DeCodeAlg	1 or 2 (as read from AvkImHdr.ImCodeVer)
StillPeriod	1

The *BitmapFormat* used to store these images can be inferred from *DecodeAlg*, since all currently supported decompression algorithms are associated with a single *BitmapFormat*.

JPEG Images

New JPEG images will be stored in files as follows:

Type	AVL_T_CIM
Sub-Type	AVL_ST_YUV_S
DeCodeAlg	129
StillPeriod	1

The only JPEG images that can be stored within a file are those for which chrominance has been subsampled 4:1 in both dimensions, and the pixel aspect ratio is 5:4.