Developments in Audio File Formats

Richard W. Dobson
The Media Technology Research Centre, Department of Mathematical Sciences
University of Bath, Bath, BA2 7AY, United Kingdom
E-mail: `masrwd@bath.ac.uk`

Of the many landmarks in the history of digital music and audio, the development of the multimedia-enabled personal computer was significant in establishing the primacy of two file formats for audio sample data. Each is associated with a particular platform – the AIFF format with the Apple Macintosh series based on the Motorola 68K (and, later, the PowerPC), and the WAVE formatwith the IBM PC and the Intel processor series. The two file formats are thus targetted at big-endian and little-endian platforms respectively, a dyad that remains important today.[1]

The similarities and differences between these two formats are significant, as not only do they demonstrate the basic requirements of a practical format, they also serve to highlight the difficulties that have arisen in the light of modern requirements.

Taking the common features first: both formats are based on the tagged chunk paradigm (derived from the original Electronic Arts "IFF 85 Standard for Interchange Format Files" specification), in which a specific chunk type serves as a container for format-specific sub-chunks. Both formats require chunks to be aligned to 16bit boundaries. Both formats are also globally *complete*, insofar as only one chunk of each type can appear in a file (this is true also of other optional chunks that can be included), and the relationship between them is well-defined – thus the interpretation of each chunk is completely defined by the format. Global completeness in a file does not necessarily indicate local completeness; two or more chunks may need to be read in order to complete the information required for rendering. Some chunks are mandatory, but a wider range of further optional chunks, for example containing production information, can also be used.

One significant difference between the formats is that WAVE requires the rendering information to precede the sample data, whereas this is not mandated in the AIFF format. Thus, strictly speaking, the WAVE format is *streamable*, whereas AIFF data may or may not be, depending on the ordering of the chunks. In the worst case, the effective *latency* (defined here as the amount of the file that must be read before rendering can start) is that of the whole SSND chunk. In the case of the WAVE format, the contents of the **fmt** chunk corresponds directly to the data structure supplied to the soundcard driver under Microsoft Windows, so that the WAVE format as a whole is *closely coupled* to the expected rendering subsystem.

Further differences arise when multi-channel audio data is considered. In the WAVE format, a two channel file is understood to contain data for the left and right speakers, but speaker assignment for further channels is undefined. We can therefore say that for multi-channel audio data, the WAVE format is incomplete, as a final decision on speaker assignment must be applied by the user or rendering device.

The AIFF specification explicitly addresses the possibility of multi-channel streams, citing examples using three, four and six channels. The problem here is *ambiguity*, as two alternative four-channel examples are given, without any disambiguating information. Such a file is therefore also not complete, as *a priori* knowledge is required to render the file correctly. Ambiguity signifies a collision between two or more existing meanings, whereas incompleteness signifies the absence of any meaning. The elimination of ambiguity is one of the central issues is designing a file format, if the data is to be independent of user intervention for rendering or processing. On the other hand, some degree of incompleteness may be unavoidable, and may indeed be worth tolerating for the sake of *flexibility* in the format[2].

Extensions to the WAVE and AIFF formats

More recently, *extensions* to WAVE and AIFF have been introduced to support floating-point audio samples. Though used by researchers for a long time, in a variety of formats, floating-point formats have gained in importance for commercial workstations (and not least with the use of processors with floating-point instructions). The different solutions found for the WAVE and AIFF formats are revealing of the consequences of the initial design of these formats.

For the WAVE format, the first field of the **fmt** chunk, **wFormatTag**, is used to identify the type of sample data: raw PCM samples, or one of a number of compressed or proprietary formats. The clear problem to be overcome is the inherent ambiguity if the sample word size is used as the type indicator – a 32bit sample can clearly be equally an integer or a floating-point number. Integer PCM samples are indicated by a value of 1 for **wFormatTag**, using the name `WAVE_FORMAT_PCM`, and the value 3 has been introduced to indicate floating-point data, using the name `WAVE_FORMAT_IEEE_FLOAT`. This format is thus commonly referred to as the 'Type-3' WAVE format, and is supported natively in Windows2000 and Windows98 Second Edition.

In contrast, the AIFF format was designed to support only uncompressed integer samples. The AIFF-C format was introduced to provide support for compressed data, and ultimately to replace AIFF. A floating-point sample type has recently been added to

[1] A big-endian version of RIFF, called RIFX, was defined at the same time, but seems to have been eclipsed by AIFF; we have not so far found any examples of its use.

[2] The question of multi-channel support under MacOS is moot, and it would appear that an ASIO or equivalent third-party driver is required for multi-channel I/O on the Macintosh.

AIFF-C, using the compression name *fl32*[3]. Significantly, AIFF-C has been defined to include a mandatory version chunk, **FVER**, allowing for the possibility of further extensions or revisions of the format. Extensibility of this kind has become an increasingly important aspect of audio file formats, on account of the rapid developments in the field, to the extent that it would now be unwise to create any new format without a version field, if the file is to serve any long-term and general purpose.

An idiomatic way to extend tagged formats, for optional information, is to define new chunks. Based on discussions among members of the music-dsp mailing list, a PEAK chunk has been defined, for use with both WAVE and AIFF, which records the value and position of the peak sample for each channel, together with a timestamp and a version field. Though developed primarily to support floating-point formats, where over- range (un-clipped) values are possible, it has use also for integer formats. It is currently supported by Soundhack (Erbe, 1992), Csound (Boulanger, 2000), and the CDP system (Endrich, 1977). The full description is available on Erbe's website[4].

A new extension to WAVE for Multi-channel data

With the development of Windows2000, Microsoft have introduced a very significant extension to the WAVE format, called `WAVE_FORMAT_EXTENSIBLE` ("WAVE-EX"). A document detailing this format is available from the Microsoft website[5]. The aspects presented in this paper are based on that document. As with the original WAVE format, `WAVE-EX` is closely coupled to the new Kernel Audio Services developed for Windows2000 (references to which can be assumed here to apply equally to Windows98 Second Edition) . A key goal of `WAVE-EX` is the explicit and unambiguous definition of speaker feeds for multi-channel streams, in order to support modern requirements, such as for DVD audio and surround sound. A total of eighteen distinct speaker feeds are defined in the specification. The first twelve of these correspond to the speaker feeds defined by the USB protocol (USB Implementers Forum, 1977); the remainder provide a reasonably complete set of feeds for height information. Speaker positions are assigned to channels using a bitmask; it is also possible to mark all channels as unassigned (a value of zero), and to assign positions to just some of the channels. Later versions of the format may define further speaker positions.

Figure 1 lists the components of the expanded `WAVE-EX` **fmt** chunk. It exploits the extensibility already present in the WAVE format to add extra fields after the standard `WAVEFORMATEX` format block, and is designed to align to a 64-bit boundary, a property that should be preserved by any custom extensions. This structure is passed by Windows2000 directly to the rendering subsystem, which maps the channels of the file to the channels supported by the selected device, performing an intelligent mixdown and/or wordsize reduction (with dithering) if necessary.

A second goal of the `WAVE-EX` format is to eliminate any remaining ambiguity with regard to the width of a sample relative to the container size. Although this distinction was made clear in the original definition of WAVE, it has become lost over time, with a variety of alternative (and, strictly speaking, non-compliant) implementations being developed. One example of a `WAVE-EX` format is a 20bit sample in a 32bit container. Thirdly, the use of a 128-bit GUID (Globally Unique IDentifer) means that independent designers can develop a custom tagged format based on `WAVE-EX` without the need to register it with Microsoft.

[3]Unfortunately, because of the somewhat un-coordinated way in which this was developed, two alternative forms of this tag are in use, using either lower-case or upper-case characters. It is likely that this will settle down soon, as support by commercial applications increases, but applications will ideally need to support both forms at least for reading, for some time.

[4]`http://shoko.calarts.edu/~tre/PeakChunk.html`

[5]`http://www.microsoft.com/hwdev/audio/multichaud.htm`

```
typedef struct {
  WAVEFORMATEX    Format;
  union {
    WORD wValidBitsPerSample;
    WORD wSamplesPerBlock;
    WORD wReserved;
  } Samples;
  DWORD dwChannelMask;
  GUID  SubFormat;
} WAVEFORMATEXTENSIBLE;
```

Figure 1: WAVE_FORMAT_EXTENSIBLE **fmt** structure

One unfortunate consequence of building `WAVE-EX` on top of the original `WAVEFORMATEX` is that **wFormatTag** is used to indicate not the sample type, but `WAVE_FORMAT_EXTENSIBLE` itself, so that it now serves two mutually exclusive tasks, introducing a potential degree of ambiguity, which may need to be resolved by a custom sub-format, as will be illustrated later. Where **wFormatTag** indicates `WAVE-EX`, the GUID then defines the sample type. Two primary GUIDs have been defined, covering PCM samples and 32bit floating-point samples, together with an initial set of compressed sample formats. A custom format can be created by defining a new GUID, and adding any new structure fields after that.

Appendices to this paper outline two examples of custom formats based on `WAVE-EX`.

Matrix formats

In addition to conventional time-domain audio streams, encoded and matrixed formats are in increasing use. A primary example is the Ambisonic B-format stream (Gerzon, 1972), which may be derived from a B-format microphone such as the Soundfield, or synthesized directly (Malham and Myatt, 1995). It is easy enough to store such a stream as a multi-channel WAVE or AIFF file, but this immediately renders the format ambiguous or incomplete, as the file is indistinguishable from a normal multi-channel file. Appendix A demonstrates a simple use of `WAVE-EX` to define a custom subtype, using the basic `WAVE-EX` structure. Two GUIDS are defined, for PCM and floating-point data. Note that the documentation advocates the use of the PEAK chunk. This can serve to indicate, for example, if a four-channel stream contains height information (Z channel). This file format is implemented in the CDP Multi-channel Toolkit[6].

Frequency-Domain formats

Frequency domain sound representations have been of the greatest importance to the computer music community for many years. Many file formats for such data have been developed anecdotally, usually tightly coupled to a particular implementation, and, in many cases, to a particular host platform. This is true, for example, of the phase vocoder (Dolson, 1983), for which many implementations are available, but all with custom and often non-portable file

[6]`http://www.bath.ac.uk/~masjpf/CDP/CDP.htm`

formats. This includes the pvoc package in the CARL distribution (Moore, 1982), the pvoc component of Csound (Karpen, 2000), and the Moore/Koonce PVC package from Princeton (Moore 1990). The Soundhack format is based on the Csound format, but writes amplitude and phase rather than amplitude and frequency. The phase vocoder in the CDP system (Fischman 1997) is based on the CARL implementation, and has used custom formats based on WAVE and AIFF to enable portability between PC and SGI platforms. The differences between the data written by these programs are very minor (different amplitude scaling factors), such that conversion programs can easily be written. In some cases byte-order is un-documented and thus host dependent, and typically the format is only documented insofar as it is possible to read the source code.

A similar situation exists for partial-tracking analysis, as described by McAulay and Quatieri (1990), with a wide range of implementations currently available. Of all these, Lemur (Fritz and Haken, 1995) is notable for including documentation on the file format. Some of these formats are not *robust*; for example, the SNDAN format (Beauchamp and Horner, 1997) begins with null-terminated text strings of arbitrary length, and programs can crash if presented with the wrong file.

The general picture therefore is of a wide variety of independently developed file formats, with varying degrees of documentation (in some cases, with no documentation at all), but in many cases differing only in low-level details. This indicates that it should be possible to design a *generic* and robust format which all such programs can support. With the high processing speeds now available on consumer machines, streaming such data in real-time is now practicable, so that the development of a streamable file format especially for phase vocoder data has become a priority need.

The ongoing SDIF initiative based at CNMAT and IRCAM (Wright, Chaudhary *et al.*, 1999) is a highly significant attempt to define open, portable and extensible file formats for a wide range of frequency domain sound representations. This has been widely documented, so we merely summarise the central aspects here, to identify the core design solutions chosen by the developers, in the context of the format attributes presented here.

The SDIF format is designed as a set of time-tagged frames containing 'matrices' of data, such as fft frames, additive partials, partial tracks, fundamental frequency estimates, and so on. The data is stored in big-endian format, and all frames are aligned to 64bit boundaries. The use of arbitrarily spaced time-stamps generally requires that the renderer implement some form of interpolation. An SDIF file can contain more than one frame or matrix type, and identical types belonging to distinct streams are disambiguated by a local stream ID. Thus a file can comprise an arbitrary collection of data in different representations. A central design criterion for SDIF is that each frame is, so far as is possible, 'stateless', which is to say that the information within it can be handled without reference to any other frame. The header of an SDIF file is minimal, and does not incorporate configuration information such as a global sample rate, based on the premise that frequency-domain data is inherently sample-rate and sample-type agnostic. In general, each frame type is intended to provide information sufficient for rendering, though the amount of information varies from one frame type to another. Thus, in the terms presented here, an SDIF file is designed to possess a high degree of local completeness.

By the same token, an SDIF file may possess a low level of global completeness, as the full contents, and global properties of each data stream, such as the highest frequency, may only be determined by scanning the whole file. An SDIF file is effectively a polymorphic or compound document, and lends itself to use as a library or database of audio data. Some combinations of matrix types (e.g sharing a common time-tag) may strongly suggest a particular application, but the final interpretation of the contents of the file will need to be made by the user, or by recourse to associated information external to the file. This seems to be the inevitable price of providing the high level of flexibility prioritized by the SDIF developers. Thus a typical implementation may depend on some direct interaction with the user. For example, in the recent implementation for Max/MSP (Wright, Dudas et al., 1999), a graphical tool is provided to enable the user to select a particular stream from the file. Also, because the highly flexible structure of an SDIF file comprises a superset of many existing file formats, while the latter may be directly converted into an SDIF file, the opposite may not be true.

Because of the much higher numeric complexity inherent in many analysis techniques, to say nothing of the possible alternative implementations, completeness must here extend to a full definition of the expected range and relationships of the data. In effect, the file format is defining a standardized implementation for a given technique, so that a file can be *application-agnostic*, and a synthesis application can correctly derive all information required for rendering from the file itself, in accordance with the documentation. In practice, this means that the documentation will need to provide either a detailed mathematical description of the data, sufficient to implement both creation and rendering, and/or a normative example open-source implementation.

A proposed phase vocoder file format

We propose a narrowly defined phase vocoder format based on `WAVE_FORMAT_EXTENSIBLE`. It illustrates the incorporation of extra format fields after the basic `WAVE-EX` structure. The format is designed to support direct rendering and streaming in real-time, by incorporating the time-domain fields of `WAVE`, and the speaker positions of `WAVE-EX`. Thus, it will support analysis and resynthesis of, for example, a 5.1 surround audio stream. Where the data is derived from analysis of a soundfile, the rendering information will be assumed to be derived directly from that file. Because the **wFormatTag** field no longer defines the sample type, and a custom GUID is used, a field is included in the PVOCDATA structure to disambiguate a 32bit sample size. The format takes care to define the expected amplitude range of each frequency bin. It is intended to be compatible with at least Csound, Soundhack, CDP, the CARL distribution and PVC, enabling these applications to exchange analysis files.

A full description, together with an example implementation based on the CARL distribution, and some conversion programs, is available from the **DREAM** website at Bath University (Dobson 2000). Comments are invited; issues currently under consideration (and which are also germane to SDIF) include the way in which analysis windows are identified, and which window types should be defined as standard. The current proposal favours the most popular representation for pvoc data as amplitude and frequency, but other representations could be included. We offer this as a prototype for any frequency-domain representation that has the potential to be streamed and rendered in real-time. Where this is not the case, `WAVE-EX` may be inappropriate, and a file format would need to be created 'from first principles'.

Appendix A: File format for Ambisonic B-Format data

This employs the basic WAVEFORMATEXTENSIBLE structure, with no added fields.

Two B-Format GUIDs are defined:

```
Integer format:
SUBTYPE_AMBISONIC_B_FORMAT_PCM
 {00000001-0721-11d3-8644-C8C1CA000000}

32bit Floating-point format:
SUBTYPE_AMBISONIC_B_FORMAT_IEEE_FLOAT
 {00000003-0721-11d3-8644-C8C1CA000000}
```

The four B-format signals are interleaved for each sample frame in the order W,X,Y,Z. If the extended six-channel B-Format is used, the U and V signals will occupy the fifth and sixth slots: W,X,Y,Z,U,V. If horizontal-only B-format is to be represented, a three or five-channel file will suffice, with signals interleaved as W,X,Y (First Order), or W,X,Y,U,V (Second-order). However, four and -six-channel files are also acceptable, with the Z channel empty. Higher-order configurations are possible in theory, but are not addressed here. A decoder program should either 'degrade gracefully', or reject formats it cannot handle.

For all B-format configurations, the dwChannelMask field should be set to zero.

Though strictly speaking an optional chunk, it is recommended that the PEAK chunk be used for all B-Format files.

Appendix B: PVOC-EX: a file format for multi-channel phase vocoder data

All information specific to the phase vocoder is contained within the PVOCDATA block. This is defined by the structure:

```
typedef struct pvoc_data {
 WORD wWordFormat;
 WORD wAnalFormat;
 WORD wSourceFormat;
 WORD wWindowType;
 DWORD nAnalysisBins;
 DWORD dwWinlen;
 DWORD dwOverlap;
 DWORD dwFrameAlign;
 float fAnalysisRate;
 float fWindowParam;
} PVOCDATA;
```

The following window types have been defined for PVOC-EX so far: PVOC_HAMMING, PVOC_HANNING, PVOC_KAISER, PVOC_RECT and PVOC_CUSTOM

The GUID defined for PVOC-EX is:

```
SUBTYPE_PVOC
{8312B9C2-2E6E-11d4-A824-DE5B96C3AB21}
```

The complete format chunk for PVOC-EX is:

```
typedef struct {
 WAVEFORMATEXTENSIBLE wxFormat;
 DWORD dwVersion;
 DWORD dwDataSize;
 PVOCDATA data;
} WAVEFORMATPVOCEX;
```

The **data** chunk comprises interleaved channels of analysis frames. Frames contain analysis bins in order Amp-Freq or Amp-Phase. Frequency values are absolute. Frames should be normalized to a peak amplitude close to 1.0. Where the PVOC_AMP_PHASE format is used, phase values are $\pm\pi$.

References

J. Beauchamp and A. Horner. Spectral modelling and timbre hybridisation programs for computer music. *Organised Sound*, 2(3): 253–258, 1997.

R. Boulanger. *The Csound Book: Tutorials in Software Synthesis and Sound Design*. MIT Press, 2000.

R. W. Dobson. Wave-ex formats. Technical report, Media Technology Research Centre, University of Bath, 2000.

M. Dolson. Musical applications of the phase vocoder. In C. Harris, editor, *Proceedings of the 1983 International Computer Music Conference*. Computer Music Assocaiation, 1983.

A. Endrich. Composers' Desktop Project: a musical imperative. *Organised Sound*, 2(1), 1997.

T. Erbe. *Soundhack users manual*. Mills College, Oakland, 1992.

R. Fischman. The Phase Vocoder: Theory and Practice. *Organised Sound*, 2(2):127–146, 1997.

K. Fritz and L. Haken. Lemur – a tool for timbre manipulation. In *Proceedings ICMC'95*, pages 158–161, 1995.

M. A. Gerzon. Periphony: With-height sound reproduction. *Journal of the Audio Engineering Society*, 21(1):2–10, 1972.

R. Karpen. *The Csound Book: Tutorials in Software Synthesis and Sound Design*, chapter 28: Csound's Phase Vocoder and Extensions. MIT Press, February 2000.

D. Malham and A. Myatt. 3-d sound spatialization using ambisonic techniques. *Computer Music Journal*, 19(4):58–70, 1995.

R. McAulay and T. Quatieri. Speech analysis/synthesis based on a sinusoidal representation. *IEEE Transactions on Acoustic,Speeach and Signal processing*, pages 744–754, 1990.

F. R. Moore. The Computer Audio Research Laboratory at UCSD. *Computer Music Journal*, 6(1):18–29, 1982.

F. R. Moore. *Elements of Computer Music*. Prentice Hall, 1990.

USB. Universal serial bus device class definition for audio devices. USB Implementers Forum, 1997.

M. Wright, A. Chaudhary, A. Freed, S. Khoury, and D. Wessel. Audio Applciaions of the Sound Description Interchange File Format. In *Proceedings of the 107th Convention*. Audio Engineering Society, 1999.

M. Wright, R. Dudas, S. Khoury, R. Wang, and D. Zicarelli. Supporting the Sound Description Interchange Format in the Max/MSP Environment. In *Proceedings ICMC'99, Beijing*, 1999.