This document was originally found at http://www.neutralzone.org/home/faqsys/docs/riff.txt, and converted to HTML by Sharky. It is suspected that several portions of the original document are incorrect, so a few corrections have been made.

RIFF WAVE (.WAV) file format

The following is taken from RIFFMCI.RTF, "Multimedia Programming Interface and Data Specification v1.0", a Windows RTF (Rich Text Format) file contained in the .zip file, RMRTF.ZRT. The original document is quite long and this constitutes pages 83-95 of the text format version (starting on roughly page 58 of the RTF version).

About the RIFF Tagged File Format

RIFF (Resource Interchange File Format) is the tagged file structure developed for multimedia resource files. The structure of a RIFF file is similar to the structure of an Electronic Arts IFF file. RIFF is not actually a file format itself (since it does not represent a specific kind of information), but its name contains the words "interchange file format" in recognition of its roots in IFF. Refer to the EA IFF definition document, EA IFF 85 Standard for Interchange Format Files, for a list of reasons to use a tagged file format.

RIFF has a counterpart, RIFX, that is used to define RIFF file formats that use the Motorola integer byte-ordering format rather than the Intel format. A RIFX file is the same as a RIFF file, except that the first four bytes are 'RIFX' instead of 'RIFF', and integer byte ordering is represented in Motorola format.

Notation Conventions

The following table lists some of the notation conventions used in this document. Further conventions and the notation for documenting RIFF forms are presented later in the document in the section "Notation for Representing Sample RIFF Files."

Notation	Description
<element label>	RIFF file element with the label "element label"
<element label: TYPE>	RIFF file element with data type "TYPE"
[<element label>]	Optional RIFF file element
<element label>...	One or more copies of the specified element
[<element label>]...	Zero or more copies of the specified element

Chunks

The basic building block of a RIFF file is called a chunk. Using C syntax, a chunk can be defined as follows:

```
typedef unsigned long DWORD;
typedef unsigned char BYTE;

typedef DWORD FOURCC;           // Four-character code

typedef FOURCC CKID;            // Four-character-code chunk identifier
typedef DWORD CKSIZE;           // 32-bit unsigned size value

typedef struct                  // Chunk structure
{
        CKID   ckID;            // Chunk type identifier
        CKSIZE ckSize;          // Chunk size field (size of ckData)
        BYTE   ckData [ckSize]; // Chunk data
} CK;
```

A FOURCC is represented as a sequence of one to four ASCII alphanumeric characters, padded on the right with blank characters (ASCII character value 32) as required, with no embedded blanks.

For example, the four-character code 'FOO' is stored as a sequence of four bytes: 'F', 'O', 'O', ' ' in ascending addresses. For quick comparisons, a four- character code may also be treated as a 32-bit number.

The three parts of the chunk are described in the following table:

Part	Description
ckID	A four-character code that identifies the representation of the chunk data data. A program reading a RIFF file can skip over any chunk whose chunk ID it doesn't recognize; it simply skips the number of bytes specified by ckSize plus the pad byte, if present.
ckSize	A 32-bit unsigned value identifying the size of ckData. This size value does not include the size of the ckID or ckSize fields or the pad byte at the end of ckData.
ckData	Binary data of fixed or variable size. The start of ckData is word-aligned with respect to the start of the RIFF file. If the chunk size is an odd number of bytes, a pad byte with value zero is written after ckData. Word aligning improves access speed (for chunks resident in memory) and maintains compatibility with EA IFF. The ckSize value does not include the pad byte.

We can represent a chunk with the following notation (in this example, the ckSize and pad byte are implicit):

```
<ckID>
(
        <ckData>
)
```

Two types of chunks, the 'LIST' and 'RIFF' chunks, may contain nested chunks, or subchunks. These special chunk types are discussed later in this document. All other chunk types store a single element of binary data in <ckData>.

Using the notation for representing a chunk, a RIFF form looks like the following:

```
RIFF
(
        <formType>
        <ck>
        ...
)
```

The first four bytes of a RIFF form make up a chunk ID with values 'R', 'I', 'F', 'F'. The ckSize field is required, but for simplicity it is omitted from the notation.

The first DWORD of chunk data in the 'RIFF' chunk (shown above as <formType>) is a four-character code value identifying the data representation, or form type, of the file. Following the form-type code is a series of subchunks. Which subchunks are present depends on the form type.

Waveform Audio File Format (WAVE)

This section describes the Waveform format, which is used to represent digitized sound.

The WAVE form is defined as follows. Programs must expect (and ignore) any unknown chunks encountered, as with all RIFF forms. However, <fmt-ck> must always occur before <wave-data>, and both of these chunks are mandatory in a WAVE file.

```
<WAVE-form> ->
        RIFF
        (
                'WAVE'
```

```
                        <fmt-ck>                // Format
                        [<fact-ck>]             // Fact chunk
                        [<cue-ck>]              // Cue points
                        [<playlist-ck>]         // Playlist
                        [<assoc-data-list>]     // Associated data list
                        <wave-data>             // Wave data
                )
```

The WAVE chunks are described in the following sections.

WAVE Format Chunk

The WAVE format chunk <fmt-ck> specifies the format of the <wave-data>. The <fmt-ck> is defined as follows:

```
<fmt-ck> ->
        fmt
        (
                <common-fields>
                <format-specific-fields>
        )

<common-fields> ->
        struct
        {
                WORD wFormatTag;          // Format category
                WORD wChannels;           // Number of channels
                DWORDdwSamplesPerSec;     // Sampling rate
                DWORDdwAvgBytesPerSec;    // For buffer estimation
                WORD wBlockAlign;         // Data block size
        }
```

The fields in the <common-fields> chunk are as follows:

Field	Description
wFormatTag	A number indicating the WAVE format category of the file. The content of the <format-specific-fields> portion of the 'fmt' chunk, and the interpretation of the waveform data, depend on this value. You must register any new WAVE format categories. See "Registering Multimedia Formats" in Chapter 1, "Overview of Multimedia Specifications," for information on registering WAVE format categories. "Wave Format Categories," following this section, lists the currently defined WAVE format categories.
wChannels	The number of channels represented in the waveform data, such as 1 for mono or 2 for stereo.
dwSamplesPerSec	The sampling rate (in samples per second) at which each channel should be played.
dwAvgBytesPerSec	The average number of bytes per second at which the waveform data should be transferred. Playback software can estimate the buffer size using this value.
wBlockAlign	The block alignment (in bytes) of the waveform data. Playback software needs to process a multiple of wBlockAlign bytes of data at a time, so the value of wBlockAlign can be used for buffer alignment.

The <format-specific-fields> consists of zero or more bytes of parameters. Which parameters occur depends on the WAVE format category-see the following section for details. Playback software should be written to allow for (and ignore) any unknown <format-specific-fields> parameters that occur at the end of this field.

WAVE Format Categories

The format category of a WAVE file is specified by the value of the wFormatTag field of the 'fmt' chunk. The representation of data in <wave-data>, and the content of the <format-specific-fields> of the 'fmt' chunk, depend on the format category.

The currently defined open non-proprietary WAVE format categories are as follows:

wFormatTag Value	Format Category
WAVE_FORMAT_PCM (0x0001)	Microsoft Pulse Code Modulation (PCM) format

The following are the registered proprietary WAVE format categories:

wFormatTag Value	Format Category
IBM_FORMAT_MULAW (0x0101)	IBM mu-law format
IBM_FORMAT_ALAW (0x0102)	IBM a-law format
IBM_FORMAT_ADPCM (0x0103)	IBM AVC Adaptive Differential Pulse Code Modulation format

The following sections describe the Microsoft WAVE_FORMAT_PCM format.

Pulse Code Modulation (PCM) Format

If the wFormatTag field of the <fmt-ck> is set to WAVE_FORMAT_PCM, then the waveform data consists of samples represented in pulse code modulation (PCM) format. For PCM waveform data, the <format-specific-fields> is defined as follows:

```
<PCM-format-specific> ->
        struct
        {
                WORD wBitsPerSample; // Sample size
        }
```

The wBitsPerSample field specifies the number of bits of data used to represent each sample of each channel. If there are multiple channels, the sample size is the same for each channel.

For PCM data, the wAvgBytesPerSec field of the 'fmt' chunk should be equal to the following formula rounded up to the next whole number:

$$wChannels \times wBitsPerSecond \times \frac{wBitsPerSample}{8}$$

Note from Sharky: The above equation is what originally appeared in this document, but it is probably incorrect. You should probably compute wAvgBytesPerSec using the formula

$$wChannels \times wSamplesPerSecond \times ceiling\left(\frac{wBitsPerSample}{8}\right)$$

The wBlockAlign field should be equal to the following formula, rounded to the next whole number:

$$wChannels \times \frac{wBitsPerSample}{8}$$

Note from Sharky: The above equation is what originally appeared in this document, but it is probably incorrect. You should probably compute wBlockAlign using the formula

$$wChannels \times ceiling\left(\frac{wBitsPerSample}{8}\right)$$

Data Packing for PCM WAVE Files

In a single-channel WAVE file, samples are stored consecutively. For stereo WAVE files, channel 0 represents the left channel, and channel 1 represents the right channel. The speaker position mapping for more than two channels is currently undefined. In multiple-channel WAVE files, samples are interleaved.

The following diagrams show the data packing for a 8-bit mono and stereo WAVE files:

Sample 1	**Sample 2**	**Sample 3**	**Sample 4**
Channel 0	Channel 0	Channel 0	Channel 0

Data Packing for 8-Bit Mono PCM

Sample 1		**Sample 2**	
Channel 0 (left)	Channel 1 (right)	Channel 0 (left)	Channel 1 (right)

Data Packing for 8-Bit Stereo PCM

The following diagrams show the data packing for 16-bit mono and stereo WAVE files:

Sample 1		**Sample 2**	
Channel 0		Channel 0	
low-order byte	high-order byte	low-order byte	high-order byte

Data Packing for 16-Bit Mono PCM

Sample 1			
Channel 0 (left)		Channel 1 (right)	
low-order byte	high-order byte	low-order byte	high-order byte

Data Packing for 16-Bit Stereo PCM

Data Format of the Samples

Each sample is contained in an integer i. The size of i is the smallest number of bytes required to contain the specified sample size. The least significant byte is stored first. The bits that represent the sample amplitude are stored in the most significant bits of i, and the remaining bits are set to zero.

For example, if the sample size (recorded in nBitsPerSample) is 12 bits, then each sample is stored in a two-byte integer. The least significant four bits of the first (least significant) byte is set to zero.

The data format and maximum and minimums values for PCM waveform samples of various sizes are as follows:

Sample Size	**Data Format**	**Maximum Value**	**Minimum Value**
One to eight bits	Unsigned integer	255 (0xFF)	0
Nine or more bits	Signed integer i	Largest positive value of i	Most negative value of i

For example, the maximum, minimum, and midpoint values for 8-bit and 16-bit PCM waveform data are as follows:

Format	**Maximum Value**	**Minimum Value**	**Midpoint Value**

8-bit PCM	255 (0xFF)	0	128 (0x80)
16-bit PCM	32767 (0x7FFF)	-32768 (-0x8000)	0

Examples of PCM WAVE Files

Example of a PCM WAVE file with 11.025 kHz sampling rate, mono, 8 bits per sample:

```
RIFF
(
        'WAVE'
        fmt
        (
                1,
                1,
                11025,
                11025,
                1,
                8
        )
        data
        (
                <wave-data>
        )
)
```

Example of a PCM WAVE file with 22.05 kHz sampling rate, stereo, 8 bits per sample:

```
RIFF
(
        'WAVE'
        fmt
        (
                1,
                2,
                22050,
                44100,
                2,
                8
        )
        data
        (
                <wave-data>
        )
)
```

Example of a PCM WAVE file with 44.1 kHz sampling rate, mono, 20 bits per sample:

```
RIFF
(
        'WAVE'
        INFO
        (
                INAM
                (
                        "O Canada"Z
                )
        )
        fmt
        (
                1,
                1,
                44100,
                132300,
                3,
                20
```

```
                        )
                        data
                        (
                                <wave-data>
                        )
                )
```

Storage of WAVE Data

The <wave-data> contains the waveform data. It is defined as follows:

```
<wave-data>  ->
{
        <data-ck> : <data-list>
}

<data-ck>    ->
        data
        (
                <wave-data>
        )

<data-list>  ->
        LIST
        (
                'wavl'
                {
                        <data-ck>    // Wave samples
                :
                        <silence-ck> // Silence
                }
                ...
        )

<silence-ck> ->
        slnt
        (
                <dwSamples:DWORD>    // Count of silent samples
        )
```

Note: The 'slnt' chunk represents silence, not necessarily a repeated zero volume or baseline sample. In 16-bit PCM data, if the last sample value played before the silence section is a 10000, then if data is still output to the D to A converter, it must maintain the 10000 value. If a zero value is used, a click may be heard at the start and end of the silence section. If play begins at a silence section, then a zero value might be used since no other information is available. A click might be created if the data following the silent section starts with a nonzero value.

FACT Chunk

The <fact-ck> fact chunk stores important information about the contents of the WAVE file. This chunk is defined as follows:

```
    <fact-ck> ->
            fact
            (
                    <dwFileSize:DWORD> // Number of samples
            )
```

The 'fact' chunk is required if the waveform data is contained in a 'wavl' LIST chunk and for all compressed audio formats. The chunk is not required for PCM files using the 'data' chunk format.

The 'fact' chunk will be expanded to include any other information required by future WAVE formats. Added fields will appear following the <dwFileSize> field. Applications can use the chunk size field to determine which fields are present.

Cue-Points Chunk

The <cue-ck> cue-points chunk identifies a series of positions in the waveform data stream. The <cue-ck> is defined as follows:

```
<cue-ck> ->
        cue
        (
                <dwCuePoints:DWORD>  // Count of cue points
                <cue-point>          // Cue-point table
                ...
        )

<cue-point> ->
        struct
        {
                DWORD  dwName;
                DWORD  dwPosition;
                FOURCC fccChunk;
                DWORD  dwChunkStart;
                DWORD  dwBlockStart;
                DWORD  dwSampleOffset;
        }
```

The <cue-point> fields are as follows:

Field	Description
dwName	Specifies the cue point name. Each <cue-point> record must have a unique dwName field.
dwPosition	Specifies the sample position of the cue point. This is the sequential sample number within the play order. See "Playlist Chunk," later in this document, for a discussion of the play order.
fccChunk	Specifies the name or chunk ID of the chunk containing the cue point.
dwChunkStart	Specifies the file position of the start of the chunk containing the cue point. This is a byte offset relative to the start of the data section of the 'wavl' LIST chunk.
dwBlockStart	Specifies the file position of the start of the block containing the position. This is a byte offset relative to the start of the data section of the 'wavl' LIST chunk.
dwSampleOffset	Specifies the sample offset of the cue point relative to the start of the block.

Examples of File Position Values

The following table describes the <cue-point> field values for a WAVE file containing multiple 'data' and 'slnt' chunks enclosed in a 'wavl' LIST chunk:

Cue Point Location	Field	Value
In a 'slnt' chunk	fccChunk	FOURCC value 'slnt'.
	dwChunkStart	File position of the 'slnt' chunk relative to the start of the data section in the 'wavl' LIST chunk.
	dwBlockStart	File position of the data section of the 'slnt' chunk relative to the start of the data section of the 'wavl' LIST chunk.

	dwSampleOffset	Sample position of the cue point relative to the start of the 'slnt' chunk.
In a PCM 'data' chunk	fccChunk	FOURCC value 'data'.
	dwChunkStart	File position of the 'data' chunk relative to the start of the data section in the 'wavl' LIST chunk.
	dwBlockStart	File position of the cue point relative to the start of the data section of the 'wavl' LIST chunk.
	dwSampleOffset	Zero value.
In a compressed 'data' chunk	fccChunk	FOURCC value 'data'.
	dwChunkStart	File position of the start of the 'data' chunk relative to the start of the data section of the 'wavl' LIST chunk.
	dwBlockStart	File position of the enclosing block relative to the start of the data section of the 'wavl' LIST chunk. The software can begin the decompression at this point.
	dwSampleOffset	Sample position of the cue point relative to the start of the block.

The following table describes the <cue-point> field values for a WAVE file containing a single 'data' chunk:

Cue Point Location	Field	Field
Within PCM data	fccChunk	FOURCC value 'data'.
	dwChunkStart	Zero value.
	dwBlockStart	Zero value.
	dwSampleOffset	Sample position of the cue point relative to the start of the 'data' chunk.
In a compressed 'data' chunk	fccChunk	FOURCC value 'data'.
	dwChunkStart	Zero value.
	dwBlockStart	File position of the enclosing block relative to the start of the 'data' chunk. The software can begin the decompression at this point.
	dwSampleOffset	Sample position of the cue point relative to the start of the block.

Playlist Chunk

The <playlist-ck> playlist chunk specifies a play order for a series of cue points. The <playlist-ck> is defined as follows:

```
<playlist-ck> ->
        plst
        (
                <dwSegments:DWORD> // Count of play segments
                <play-segment>...  // Play-segment table
        )

<play-segment> ->
        struct
        {
                DWORD dwName;
                DWORD dwLength;
                DWORD dwLoops;
        }
```

The <play-segment> fields are as follows:

Field	Description
dwName	Specifies the cue point name. This value must match one of the names listed in the <cue-ck>

	cue-point table.
dwLength	Specifies the length of the section in samples.
dwLoops	Specifies the number of times to play the section.

Associated Data Chunk

The <assoc-data-list> associated data list provides the ability to attach information like labels to sections of the waveform data stream. The <assoc-data-list> is defined as follows:

```
<assoc-data-list> ->
        LIST
        (
                'adtl'
                <labl-ck> // Label
                <note-ck> // Note
                <ltxt-ck> // Text with data length
                <file-ck> // Media file
        )

<labl-ck> ->
        labl
        (
                <dwName:DWORD>
                <data:ZSTR>
        )

<note-ck> ->
        note
        (
                <dwName:DWORD>
                <data:ZSTR>
        )

<ltxt-ck> ->
        ltxt
        (
                <dwName:DWORD>
                <dwSampleLength:DWORD>
                <dwPurpose:DWORD>
                <wCountry:WORD>
                <wLanguage:WORD>
                <wDialect:WORD>
                <wCodePage:WORD>
                <data:BYTE>...
        )

<file-ck> ->
        file
        (
                <dwName:DWORD>
                <dwMedType:DWORD>
                <fileData:BYTE>...
        )
```

Label and Note Information

The 'labl' and 'note' chunks have similar fields. The 'labl' chunk contains a label, or title, to associate with a cue point. The 'note' chunk contains comment text for a cue point. The fields are as follows:

Field	Description
dwName	Specifies the cue point name. This value must match one of the names listed in the <cue-ck> cue-point table.

data	Specifies a NULL-terminated string containing a text label (for the 'labl' chunk) or comment text (for the 'note' chunk).

Text with Data Length Information

The "ltxt" chunk contains text that is associated with a data segment of specific length. The chunk fields are as follows:

Field	Description
dwName	Specifies the cue point name. This value must match one of the names listed in the <cue-ck> cue-point table.
dwSampleLength	Specifies the number of samples in the segment of waveform data.
dwPurpose	Specifies the type or purpose of the text. For example, dwPurpose can specify a FOURCC code like 'scrp' for script text or 'capt' for close- caption text.
wCountry	Specifies the country code for the text. See "Country Codes" in Chapter 2, "Resource Interchange File Format," for a current list of country codes.
wLanguage, wDialect	Specify the language and dialect codes for the text. See "Language and Dialect Codes" in Chapter 2, "Resource Interchange File Format," for a current list of language and dialect codes.
wCodePage	Specifies the code page for the text.

Embedded File Information

The 'file' chunk contains information described in other file formats (for example, an 'RDIB' file or an ASCII text file). The chunk fields are as follows:

Field	Description
dwName	Specifies the cue point name. This value must match one of the names listed in the <cue-ck> cue-point table.
dwMedType	Specifies the file type contained in the fileData field. If the fileData section contains a RIFF form, the dwMedType field is the same as the RIFF form type for the file. This field can contain a zero value.
fileData	Contains the media file.