WAVファイルをテキストにダンプして内容を可視化する

2017/09/19 作成

WAVEファイルの構造を調査している過程で作ったやっつけツール。
~~※このソースをベースに、dataチャンクの取り出しと、再構成(再出力)をサポートしたクラスを作り~~VOICEROID+EXを制御するDLL~~に組み込む予定。~~

構造を調べるだけなら、GUIで利用可能なRIFFPadがよいんじゃないかなと思います。 RIFFPad – RIFF File Viewer

ダウンロード

2017-09-19版
.NET Framework 4.5.2で作成、.NET Framework 4.6 でも動くと思う

注意

- 暗号化された形式には対応できていません。てかそんなファイルが入手できません。
- 知らない形式のChunkはダンプだけになります。資料が入手できないのではどうしようもないです。

参照情報

- Resource Interchange File Format Services
- Wav file format
- Hatada's Home Page - 音響信号処理
- サウンド・プレーヤ開発状況
- WAVE ファイルフォーマット
- RIFFファイルフォーマット
- Microsoft Multimedia Standards Update(PDF)
- Multimedia Programming Interface and Data Specifications 1.0(PDF)
- d3v2.3.0 - ID3.org
- ID3 タグバージョン2.3.0

日本語の情報はほとんど fmtチャンクとdataチャンクの説明で終わっていて、LISTチャンクやCueチャンクの情報を探すのは困難。

使い方

アーカイブを解凍して C:\work フォルダに wavedump.exe をコピーしたのであれば以下のコマンドラインでWAVファイル hoge.wav の内容をテキストファイルにダンプします。

```
C:\work> wavedump hoge.wav
```

ダンプした結果は標準出力に出力されます。ファイルにダンプ結果を出力するなら -o オプションを使います。

```
C:\work> wavedump -o hoge.txt hoge.wav
```

ダンプ結果が出力された hoge.txt はこんな感じになります。

```
======== ================================================
00000000 52 49 46 46              > Chunk ID : "RIFF"
00000004 22 A7 00 00              > Chunk Size : "42786"
00000008 57 41 56 45              > Form Type : "WAVE"
======== ================================================
0000000C 66 6D 74 20              > Chunk ID : "fmt "
00000010 10 00 00 00              > Chunk Size : "16"
00000014 01 00                    > Format : "1"
00000016 01 00                    > Channels : "1"
00000018 22 56 00 00              > SampleRate : "22050"
0000001C 44 AC 00 00              > ByteRate : "44100"
00000020 02 00                    > BlockAlign : "2"
00000022 10 00                    > BitPerSample : "16"
======== ================================================
00000024 64 61 74 61              > Chunk ID : "data"
00000028 FE A6 00 00              > Chunk Size : "42750"
0000002C 68 00 33 00 40 00 68 00 3E 00 C7 FF 93 FF 90 FF
0000003C 7E FF 3D FF EF FE D2 FE 0E FF 47 FF 35 FF FB FE
0000004C D4 FE C6 FE C2 FE CB FE 9C FE 6B FE 8C FE CF FE
0000005C A4 FE F5 FD B3 FD F2 FD F6 FD 62 FD 05 FD 1B FD
   -- 中略 --
0000A6EC 07 00 00 00 00 00 11 00 26 00 26 00 20 00 18 00
0000A6FC 0F 00 FE FF F9 FF F9 FF EF FF F2 FF FD FF 06 00
0000A70C 0B 00 11 00 10 00 0D 00 07 00 02 00 FF FF FD FF
0000A71C FD FF FD FF FE FF 00 00 00 00 00 00 00 00
-- Chunk Index --
00000000,RIFF,42786
0000000C,fmt ,16
00000024,data,42750
```

上記はWAVファイルとしては最低限のチャンク構造です。
fmtチャンクより、非圧縮PCM(Format=1)の量子化ビット数16(16bit)モノラル(Channels=1)サンプリングレート22.05kHz1秒当たりのデータ量は44100(22050×2)バイト、とわかります。

チャンク(Chunk)の説明

WAVEファイルはRIFF形式に従って、チャンク(Chunk)と呼ぶデータブロックの集合でできています。参照情報に挙げたリンクから確認できるでしょう。

- RIFFチャンク(FormType:WAVE)の内包チャンクに最低限fmtチャンクとdataチャンクが必要にな

ります。

- fmtチャンクの内容でWAVファイルのデータ形式を定義□dataチャンクで実際の音声データを定義しています。
- dataチャンクから音声データを抜き出すようなプログラムを作る際には、手を抜いてfmtチャンク□dataチャンクが連続する前提でプログラムを書かない方が良いです。
  fmtチャンクとdataチャンクの間にfactチャンクが存在する例があり(非圧縮なPCMデータであればfactチャンクは存在しないしあってもスキップすればいい)、決め打ちで処理すると痛い目を見ます。
- LISTチャンクには著作権情報や作成時のプログラムに関する情報が入っていることがあります。
  下記は、LISTチャンクのサブチャンクであるISFTチャンクに“ Coderium SoundEngine 5.10”とあり、おそらく http://soundengine.jp/ に説明のあるSoundEngine v5を使って作成されたのではないかと推測ができます。

```
======== ================================================
00000000 52 49 46 46              > Chunk ID : "RIFF"
00000004 52 EE 17 00              > Chunk Size : "1568338"
00000008 57 41 56 45              > Form Type : "WAVE"
======== ================================================
0000000C 66 6D 74 20              > Chunk ID : "fmt "
00000010 10 00 00 00              > Chunk Size : "16"
00000014 01 00                    > Format : "1"
00000016 01 00                    > Channels : "1"
00000018 00 77 01 00              > SampleRate : "96000"
0000001C 00 65 04 00              > ByteRate : "288000"
00000020 03 00                    > BlockAlign : "3"
00000022 18 00                    > BitPerSample : "24"
======== ================================================
00000024 64 61 74 61              > Chunk ID : "data"
00000028 00 EE 17 00              > Chunk Size : "1568256"
0000002C 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000003C 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000004C 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
    -- 中略 --
0017EDFC 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0017EE0C 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0017EE1C 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
======== ================================================
0017EE2C 4C 49 53 54              > Chunk ID : "LIST"
0017EE30 26 00 00 00              > Chunk Size : "38"
0017EE34 49 4E 46 4F              > ListType : "INFO"
-------- ------------------------------------------------
0017EE38 49 53 46 54              > Chunk ID : "ISFT"
0017EE3C 19 00 00 00              > Chunk Size : "25"
0017EE40 43 6F 64 65 72 69 75 6D  > TEXT : "Coderium SoundEngine 5.10 "
0017EE48 20 53 6F 75 6E 64 45 6E
0017EE50 67 69 6E 65 20 35 2E 31
0017EE58 30 00
-- Chunk Index --
00000000,RIFF,1568338
0000000C,fmt ,16
```

```
00000024,data,1568256
0017EE2C,LIST,38
0017EE38,ISFT,25
```

- このチャンクには色々なものがあって、メジャーなもの以外はその構造がわからないものがほとんど。実際にファイルをダンプして初めて存在に気が付いたり。有名どころのAudacityで作成したWavファイルには“id3 “チャンクが入っていて、まんまMP3ファイルで使われているID3タグのデータが含められていました※wavedumpではある程度ID3タグのパースも行います。
- 他にも、公開されているWavファイルをダンプするとLISTチャンクのサブチャンクにこれまた知らない形式のチャンクが見つかったりします。LIST(ListType:LINK)チャンクのサブチャンクにPATHチャンクなるものを見つけた時はどうにも情報が見つけられずパース処理組み込みを断念しました。

```
======== ================================================
00000000 52 49 46 46              > Chunk ID : "RIFF"
00000004 FE D0 14 00              > Chunk Size : "1364222"
00000008 57 41 56 45              > Form Type : "WAVE"
======== ================================================
0000000C 66 6D 74 20              > Chunk ID : "fmt "
00000010 10 00 00 00              > Chunk Size : "16"
00000014 01 00                    > Format : "1"
00000016 01 00                    > Channels : "1"
00000018 44 AC 00 00              > SampleRate : "44100"
0000001C 88 58 01 00              > ByteRate : "88200"
00000020 02 00                    > BlockAlign : "2"
00000022 10 00                    > BitPerSample : "16"
======== ================================================
00000024 64 61 74 61              > Chunk ID : "data"
00000028 CE C9 14 00              > Chunk Size : "1362382"
0000002C 81 00 9F 01 AE 02 A2 03 9E 04 6A 05 3A 06 31 07
0000003C DF 07 7E 08 2A 09 79 09 BD 09 C2 09 C2 09 C3 09
    -- 中略 --
0014C9DC D4 02 B2 02 9F 02 92 02 74 02 69 02 6F 02 1D 02
0014C9EC 9A 01 E2 00 C3 FF 90 FE 72 FD 86 FC E0 FB
======== ================================================
0014C9FA 53 79 4C 70              > Chunk ID : "SyLp"
0014C9FE B8 00 00 00              > Chunk Size : "184"
0014CA02 01 00 00 00 00 00 00 00 9A 99 99 99 99 99 40 40
0014CA12 0E 2D B2 9D EF 1F 60 40 00 00 00 00 00 00 00 00
    -- 中略 --
0014CAA2 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0014CAB2 00 00 00 00 00 00 00 00
======== ================================================
0014CABA 43 72 38 72              > Chunk ID : "Cr8r"
0014CABE 54 00 00 00              > Chunk Size : "84"
0014CAC2 BE EF CA FE 00 00 00 54 00 01 00 00 46 58 54 43
0014CAD2 43 72 38 72 2E 73 65 73 00 FF FF FF 36 C3 66 74
0014CAE2 45 FF D0 77 2F 45 50 20 00 00 00 00 20 D8 12 00
0014CAF2 12 01 00 00 41 75 64 69 74 69 6F 6E 00 95 CF 77
0014CB02 00 00 03 00 02 00 00 00 20 D8 12 00 8E 95 CF 77
0014CB12 12 01 00 00
```

```
======== ================================================
0014CB16 72 38 72 43              > Chunk ID : "r8rC"
0014CB1A 54 00 00 00              > Chunk Size : "84"
0014CB1E BE EF CA FE 00 00 00 54 00 01 00 00 46 58 54 43
0014CB2E 43 72 38 72 2E 73 65 73 00 FF FF FF 36 C3 66 74
0014CB3E 45 FF D0 77 2F 45 50 20 00 00 00 00 20 D8 12 00
0014CB4E 12 01 00 00 41 75 64 69 74 69 6F 6E 00 95 CF 77
0014CB5E 00 00 03 00 02 00 00 00 20 D8 12 00 8E 95 CF 77
0014CB6E 12 01 00 00
======== ================================================
0014CB72 4C 49 53 54              > Chunk ID : "LIST"
0014CB76 8C 05 00 00              > Chunk Size : "1420"
0014CB7A 4C 49 4E 4B              > ListType : "LINK"
-------- ------------------------------------------------
0014CB7E 50 41 54 48              > Chunk ID : "PATH"
0014CB82 80 05 00 00              > Chunk Size : "1408"
0014CB86 45 3A 5C 57 41 56 5C 53 45 5C 73 79 6E 74 68 5C
0014CB96 83 7B 81 5B 83 67 5C 83 7D 83 8B 83 60 2E 73 65
    -- 中略 --
0014D0E6 60 9A 17 00 9A CC D1 77 81 02 00 00 48 A1 57 00
0014D0F6 5C A1 57 00 00 00 00 00 00 00 00 00 00 00 00 00
-- Chunk Index --
00000000,RIFF,1364222
0000000C,fmt ,16
00000024,data,1362382
0014C9FA,SyLp,184
0014CABA,Cr8r,84
0014CB16,r8rC,84
0014CB72,LIST,1420
0014CB7E,PATH,1408
```

- ちょっと怪しめのところにあったWAVファイルは、ZIPチャンクが存在していてデータとして圧縮されたdataチャンクが組み込まれていて□dataチャンクと入れ替え可能なものがありました。擬装されたWavファイル、とでも呼べばいいのかな？

ソース

チャンクサイズ分だけメモリを確保してパースを行うので、それほどメモリ効率が良いものではありません。
長い時間記録したWAVEファイルはdataチャンクだけで大変メモリを消費する事になるでしょう。

wavedump.cs

```cs
using System;
using System.Collections.Generic;
using System.IO;
using System.Text;

namespace wavcontrolsample
```

```
{
    public class wavedump
    {
        static void Main(string[] args)
        {
            WaveFileParser wfp;
            string wavefile;

            if (args.Length == 1)
            {
                wfp = new WaveFileParser();
                wavefile = args[0];
            }
            else if (args.Length == 3)
            {
                if (args[0] == "-o")
                {
                    wfp = new WaveFileParser(args[1]);
                    wavefile = args[2];
                }
                else
                {
                    help();
                    return;
                }
            }
            else
            {
                help();
                return;
            }

            wfp.ParseWavFile(wavefile);
            wfp.ListChunkIndex();
            wfp.FlushWriteLine();
        }

        static void help()
        {
            Console.WriteLine("usage: wavdump [-o dumpfile] wavefile");
            Console.WriteLine("");
        }
    }

    public class WaveFileParser
    {
        private List<string> ChunkLocation = new List<string>();
        private StreamWriter sw;

        public WaveFileParser()
        {
```

```
            sw = new StreamWriter(Console.OpenStandardOutput(),
Encoding.Default);
        }
        public WaveFileParser(string outputFilePath)
        {
            sw = new StreamWriter(outputFilePath, false,
Encoding.Default);
        }
        public void FlushWriteLine()
        {
            sw.Flush();
        }
        public void ListChunkIndex()
        {
            WriteLine("-- Chunk Index --");
            foreach (string arg in ChunkLocation)
            {
                WriteLine(arg);
            }
        }
        public void ParseWavFile(string wavFileName)
        {
            byte[] buff = new byte[4];
            UInt32 size;

            string ChunkID;
            UInt32 ChunkSize;
            string FormType;
            UInt32 Location = 0;

            using (FileStream fs = new FileStream(wavFileName,
FileMode.Open))
            {
                WriteLine("========
==============================================");

                // RIFFチャンク□WAVEフォームでないなら終了
                fs.Read(buff, 0, 4);
                ChunkID = Encoding.ASCII.GetString(buff);
                printData(Location, buff, "Chunk ID", ChunkID);
                Location += 4;

                fs.Read(buff, 0, 4);
                ChunkSize = (UInt32)((buff[3] << 24) + (buff[2] << 16)
+ (buff[1] << 8) + buff[0]);
                printData(Location, buff, "Chunk Size", ChunkSize);
                Location += 4;

                fs.Read(buff, 0, 4);
                FormType = Encoding.ASCII.GetString(buff);
                printData(Location, buff, "Form Type", FormType);
```

```
                Location += 4;

                if (ChunkID != "RIFF") throw new
Exception(string.Format("Can't read except \"RIFF\" format. this is
\"{0}\" format.", ChunkID));
                if (FormType != "WAVE") throw new
Exception(string.Format("Can't read except \"WAVE\" form. this is
\"{0}\" form.", FormType));

                size = (UInt32)(ChunkSize - 4);
                if ((size % 2) == 1) size += 1;

                ChunkLocation.Add(string.Format("{0:X08},{1},{2}", 0,
ChunkID, ChunkSize));

                //子チャンクの読み取り

                UInt32 size2 = 0;
                UInt32 paddingSize;
                byte[] subBuff;

                while (size2 < size)
                {
                    if (4 > fs.Read(buff, 0, 4)) break;

                    WriteLine("========
================================================");
                    ChunkID = Encoding.ASCII.GetString(buff);
                    printData(Location, buff, "Chunk ID", ChunkID);
                    Location += 4;

                    fs.Read(buff, 0, 4);
                    ChunkSize = (UInt32)((buff[3] << 24) + (buff[2] <<
16) + (buff[1] << 8) + buff[0]);
                    printData(Location, buff, "Chunk Size", ChunkSize);
                    Location += 4;

                    paddingSize = ((ChunkSize % 2) == 1) ? ChunkSize +
1 : ChunkSize;

                    ChunkLocation.Add(string.Format("{0:X08},{1},{2}",
Location - 8, ChunkID, ChunkSize));

                    subBuff = new byte[paddingSize];
                    fs.Read(subBuff, 0, (Int32)ChunkSize);

                    switch (ChunkID)
                    {
                        case "fmt ":
                            ParseFmt(Location, subBuff, ChunkSize);
                            break;
```

```
                        case "data":
                            printData(Location, subBuff);
                            break;

                        case "LIST":
                            ParseList(Location, subBuff);
                            break;

                        case "id3 ":
                            ParseID3(Location, subBuff);
                            break;

                        case "fact":
                            break;

                        case "cue ":
                            ParseCue(Location, subBuff);
                            break;

                        default:
                            printData(Location, subBuff);
                            break;
                    }

                    Location += paddingSize;
                    size2 += (8 + paddingSize);
                }
            }
        }

        private void ParseFmt(UInt32 Location, byte[] data, UInt32
chunkSize)
        {
            int pos = 0;
            byte[] buff4 = new byte[4];
            byte[] buff2 = new byte[2];

            Array.Copy(data, pos, buff2, 0, 2);
            printData(Location + (UInt32)pos, buff2, "Format",
Size2Calc(buff2));
            pos += 2;

            Array.Copy(data, pos, buff2, 0, 2);
            printData(Location + (UInt32)pos, buff2, "Channels",
Size2Calc(buff2));
            pos += 2;

            Array.Copy(data, pos, buff4, 0, 4);
            printData(Location + (UInt32)pos, buff4, "SampleRate",
Size4Calc(buff4));
```

```
            pos += 4;

            Array.Copy(data, pos, buff4, 0, 4);
            printData(Location + (UInt32)pos, buff4, "ByteRate",
Size4Calc(buff4));
            pos += 4;

            Array.Copy(data, pos, buff2, 0, 2);
            printData(Location + (UInt32)pos, buff2, "BlockAlign",
Size2Calc(buff2));
            pos += 2;

            Array.Copy(data, pos, buff2, 0, 2);
            printData(Location + (UInt32)pos, buff2, "BitPerSample",
Size2Calc(buff2));
            pos += 2;

            if (chunkSize > 16)
            {
                Array.Copy(data, pos, buff2, 0, 2);
                UInt16 ExtendedSize = Size2Calc(buff2);
                printData(Location + (UInt32)pos, buff2,
"ExtendedSize", ExtendedSize);
                pos += 2;

                byte[] ExtendedData = new byte[ExtendedSize];
                Array.Copy(data, pos, ExtendedData, 0, ExtendedSize);
                printData(Location + (UInt32)pos, ExtendedData,
"ExtendedData", ExtendedSize+" byte");
                //pos += ExtendedSize;
            }
        }
        private void ParseList(UInt32 Location, byte[] data)
        {
            int pos = 0;
            string ChunkID;
            UInt32 ChunkSize;
            UInt32 paddingSize;
            byte[] buff4 = new byte[4];
            byte[] buff2 = new byte[2];
            byte[] subBuff;
            byte[] subBuff2;

            Array.Copy(data, pos, buff4, 0, 4);
            printData(Location + (UInt32)pos, buff4, "ListType",
Encoding.ASCII.GetString(buff4));
            pos += 4;

            while (pos < data.Length)
            {
                WriteLine("-------- ----------------------------------
```

```
-------------");

                Array.Copy(data, pos, buff4, 0, 4);
                ChunkID = Encoding.ASCII.GetString(buff4);
                printData(Location + (UInt32)pos, buff4, "Chunk ID",
ChunkID);
                pos += 4;

                Array.Copy(data, pos, buff4, 0, 4);
                ChunkSize = Size4Calc(buff4);
                printData(Location + (UInt32)pos, buff4, "Chunk Size",
ChunkSize);
                pos += 4;

                ChunkLocation.Add(string.Format("{0:X08},{1},{2}",
Location + pos - 8, ChunkID, ChunkSize));

                paddingSize = ((ChunkSize % 2) == 1) ? ChunkSize + 1 :
ChunkSize;

                subBuff = new byte[paddingSize];
                Array.Copy(data, pos, subBuff, 0, ChunkSize);

                switch (ChunkID)
                {
                    // INFO Chunk
                    case "IARL":
                    case "IART":
                    case "ICMS":
                    case "ICMT":
                    case "ICOT":
                    case "ICRD":
                    case "ICRP":
                    case "IDIM":
                    case "IDPI":
                    case "IENG":
                    case "IGNR":
                    case "IKEY":
                    case "ILGT":
                    case "IMED":
                    case "INAM":
                    case "IPLT":
                    case "IPRD":
                    case "ISBJ":
                    case "ISFT":
                    case "ISHP":
                    case "ISRC":
                    case "ISRF":
                    case "ITCH":
                    case "ITRK":   // Application original?
                        printData(Location + 0 + (UInt32)pos, subBuff,
```

```
"TEXT", Encoding.ASCII.GetString(subBuff));
                        break;

                    // adtl Chunk
                    case "labl":
                    case "note":
                        Array.Copy(subBuff, 0, buff4, 0, 4);
                        printData(Location + 0 + (UInt32)pos, subBuff,
"Cue Point Name", Size4Calc(buff4));

                        subBuff2 = new byte[subBuff.Length - 4];
                        Array.Copy(subBuff, 4, subBuff2, 0,
subBuff.Length - 4);
                        printData(Location + 4 + (UInt32)pos, subBuff2,
"TEXT", Encoding.ASCII.GetString(subBuff2));
                        break;

                    case "file":
                        Array.Copy(subBuff, 0, buff4, 0, 4);
                        printData(Location + 0 + (UInt32)pos, subBuff,
"Cue Point Name", Size4Calc(buff4));

                        Array.Copy(subBuff, 4, buff4, 0, 4);
                        printData(Location + 4 + (UInt32)pos, buff2,
"MedType");

                        subBuff2 = new byte[subBuff.Length - 8];
                        Array.Copy(subBuff, 8, subBuff2, 0,
subBuff.Length - 8);
                        printData(Location + 8 + (UInt32)pos, subBuff2,
"file data");
                        break;

                    case "ltxt":
                        Array.Copy(subBuff, 0, buff4, 0, 4);
                        printData(Location + 0 + (UInt32)pos, subBuff,
"Cue Point Name", Size4Calc(buff4));

                        Array.Copy(subBuff, 4, buff4, 0, 4);
                        printData(Location + 4 + (UInt32)pos, subBuff,
"Sample Length", Size4Calc(buff4));

                        Array.Copy(subBuff, 8, buff4, 0, 4);
                        printData(Location + 8 + (UInt32)pos, buff4,
"Purpose", Encoding.ASCII.GetString(buff4));

                        Array.Copy(subBuff, 12, buff2, 0, 2);
                        printData(Location + 12 + (UInt32)pos, buff2,
"Country", Encoding.ASCII.GetString(buff2));

                        Array.Copy(subBuff, 14, buff2, 0, 2);
```

```
                        printData(Location + 14 + (UInt32)pos, buff2,
"Language", Encoding.ASCII.GetString(buff2));

                        Array.Copy(subBuff, 16, buff2, 0, 2);
                        printData(Location + 16 + (UInt32)pos, buff2,
"Dialect");

                        Array.Copy(subBuff, 18, buff2, 0, 2);
                        printData(Location + 18 + (UInt32)pos, buff2,
"CodePage", Size2Calc(buff2));

                        if (ChunkSize > 20)
                        {
                            subBuff2 = new byte[subBuff.Length - 20];
                            Array.Copy(subBuff, 20, subBuff2, 0,
subBuff.Length - 20);
                            printData(Location + 20 + (UInt32)pos,
subBuff2, "TEXT", Encoding.ASCII.GetString(subBuff2));
                        }
                        break;

                    default:
                        printData(Location + (UInt32)pos, subBuff);
                        break;
                }

                pos += (Int32)paddingSize;
            }

        }
        private void ParseFact(UInt32 Location, byte[] data)
        {
            int pos = 0;
            byte[] buff4 = new byte[4];

            Array.Copy(data, pos + 0, buff4, 0, 4);
            printData(Location + (UInt32)pos, buff4, "Samples",
Size4Calc(buff4));
            pos += 4;
        }
        private void ParseCue(UInt32 Location, byte[] data)
        {
            int pos = 0;
            UInt32 CuePointCount;
            byte[] buff4 = new byte[4];

            Array.Copy(data, pos, buff4, 0, 4);
            CuePointCount = Size4Calc(buff4);
            printData(Location + (UInt32)pos, buff4, "CuePointCount",
CuePointCount);
            pos += 4;
```

```
            if (CuePointCount != 0)
            {
                for (UInt32 idx = 0; idx < CuePointCount; idx++)
                {
                    WriteLine("-------- ------------------------------
-----------------");

                    Array.Copy(data, pos, buff4, 0, 4);
                    printData(Location + (UInt32)pos, buff4,
"CueNumer", Size4Calc(buff4));
                    pos += 4;

                    Array.Copy(data, pos, buff4, 0, 4);
                    printData(Location + (UInt32)pos, buff4,
"Position", Size4Calc(buff4));
                    pos += 4;

                    Array.Copy(data, pos, buff4, 0, 4);
                    printData(Location + (UInt32)pos, buff4, "Chunk",
Encoding.ASCII.GetString(buff4));
                    pos += 4;

                    Array.Copy(data, pos, buff4, 0, 4);
                    printData(Location + (UInt32)pos, buff4,
"ChunkStart", Size4Calc(buff4));
                    pos += 4;

                    Array.Copy(data, pos, buff4, 0, 4);
                    printData(Location + (UInt32)pos, buff4,
"BlockStart", Size4Calc(buff4));
                    pos += 4;

                    Array.Copy(data, pos, buff4, 0, 4);
                    printData(Location + (UInt32)pos, buff4,
"SampleOffset", Size4Calc(buff4));
                    pos += 4;
                }
            }
        }
        private void ParseID3(UInt32 Location, byte[] data)
        {
            int pos = 0;
            string identifier;
            UInt32 frameSize;
            string frameId;
            string dataText;

            byte[] buff4 = new byte[4];
            byte[] buff3 = new byte[3];
            byte[] buff2 = new byte[2];
```

```
            byte[] buff1 = new byte[1];
            byte[] flags = new byte[1];
            byte[] subBuff;
            string flagStr;


            /////////// ID3 ヘッダのパース

            Array.Copy(data, pos, buff3, 0, 3);
            identifier = Encoding.ASCII.GetString(buff3);
            printData(Location + (UInt32)pos, buff3, "IDENTIFIRE",
identifier);
            pos += 3;

            if ("ID3" != identifier)
            {
                subBuff = new byte[data.Length - 3];
                printData(Location + (UInt32)pos, subBuff);
                return;
            }

            Array.Copy(data, pos, buff2, 0, 2);
            printData(Location + (UInt32)pos, buff2, "VERSION",
string.Format("{0:D02}-{1:D02}", buff2[0], buff2[1]));
            pos += 2;

            Array.Copy(data, pos, flags, 0, 1);
            flagStr = ((flags[0] & 0x80) != 0) ? "Unsynchronisation, "
: "";
            flagStr += ((flags[0] & 0x40) != 0) ? "ExtentedHeader, "
: "";
            flagStr += ((flags[0] & 0x20) != 0) ? "Experimental" : "";
            flagStr = flagStr.Trim(',');
            printData(Location + (UInt32)pos, flags, "FLAGS", flagStr);
            pos++;

            Array.Copy(data, pos, buff4, 0, 4);
            printData(Location + (UInt32)pos, buff4, "Frame Size",
ID3Size4Calc(buff4));
            pos += 4;

            //拡張ヘッダがある
            if ((flags[0] & 0x40) != 0)
            {
                Array.Copy(data, pos, buff4, 0, 4);
                printData(Location + (UInt32)pos, buff4, "Extended
Header Size", ID3Size4Calc(buff4));
                pos += 4;

                Array.Copy(data, pos, buff2, 0, 2);
                flagStr = ((buff2[0] & 0x80) != 0) ? "CRC" : "";
```

```
                printData(Location + (UInt32)pos, buff2, "Extened
Header Flags", flagStr);
                pos += 2;

                Array.Copy(data, pos, buff4, 0, 4);
                printData(Location + (UInt32)pos, buff4, "Extended
Header Size", ID3Size4Calc(buff4));
                pos += 4;

                Array.Copy(data, pos, buff4, 0, 4);
                printData(Location + (UInt32)pos, buff4, "Extended
Padding Size", ID3Size4Calc(buff4));
                pos += 4;

                // CRCがある場合
                if ((buff2[0] & 0x80) != 0)
                {
                    Array.Copy(data, pos, buff4, 0, 4);
                    printData(Location + (UInt32)pos, buff4, "Extended
Header CRC");
                    pos += 4;
                }
            }

            /////////// ID3 フレームのパース

            while (pos < data.Length)
            {
                if (10 > (data.Length - pos)) break; //フレームヘッダサイ
ズは10バイト

                WriteLine("-------- ----------------------------------
-------------");

                Array.Copy(data, pos, buff4, 0, 4);
                frameId = Encoding.ASCII.GetString(buff4);
                printData(Location + (UInt32)pos, buff4, "FrameID",
frameId);
                pos += 4;

                Array.Copy(data, pos, buff4, 0, 4);
                frameSize = ID3Size4Calc(buff4);
                printData(Location + (UInt32)pos, buff4, "Frame Size",
frameSize);
                pos += 4;

                ChunkLocation.Add(string.Format("{0:X08},{1},{2}",
Location + pos - 8, frameId, frameSize));

                Array.Copy(data, pos, buff2, 0, 2);
                printData(Location + (UInt32)pos, buff2, "Frame
```

```
Flags");
                pos += 2;

                if (frameId.StartsWith("T") || ("WXXX" == frameId))
                {
                    Array.Copy(data, pos, buff1, 0, 1);
                    printData(Location + (UInt32)pos, buff1,
"TextEncoding");
                    pos ++;

                    subBuff = new byte[frameSize - 1];
                    Array.Copy(data, pos, subBuff, 0, frameSize - 1);
                    if (buff1[0] == 0x00)
                    {
                        dataText = Encoding.ASCII.GetString(subBuff);
                    }
                    else
                    {
                        dataText = Encoding.Unicode.GetString(subBuff);
                    }
                    dataText = dataText.TrimEnd('\0');
                    printData(Location + (UInt32)pos, subBuff, "text",
dataText);
                    pos += (Int32)(frameSize - 1);
                }
                else if (frameId.StartsWith("W"))
                {
                    subBuff = new byte[frameSize];
                    Array.Copy(data, pos, subBuff, 0, frameSize);
                    dataText = Encoding.ASCII.GetString(subBuff);
                    dataText = dataText.TrimEnd('\0');
                    printData(Location + (UInt32)pos, subBuff, "url",
dataText);
                    pos += (Int32)(frameSize);
                }
                else
                {
                    switch (frameId)
                    {
                        case "COMM":
                            Array.Copy(data, pos, buff1, 0, 1);
                            printData(Location + (UInt32)pos, buff1,
"TextEncoding");
                            pos++;

                            Array.Copy(data, pos, buff3, 0, 3);
                            dataText = Encoding.ASCII.GetString(buff3);
                            dataText = dataText.TrimEnd('\0');
                            printData(Location + (UInt32)pos, buff3,
"Language", dataText);
                            pos += 3;
```

```
                            subBuff = new byte[frameSize - 4];
                            Array.Copy(data, pos, subBuff, 0, frameSize
- 4);
                            if (buff1[0] == 0x00)
                            {
                                dataText =
Encoding.ASCII.GetString(subBuff);
                            }
                            else
                            {
                                dataText =
Encoding.Unicode.GetString(subBuff);
                            }
                            dataText = dataText.TrimEnd('\0');
                            printData(Location + (UInt32)pos, subBuff,
"text", dataText);
                            pos += (Int32)(frameSize - 4);
                            break;

                        default:
                            subBuff = new byte[frameSize];
                            Array.Copy(data, pos, subBuff, 0,
frameSize);
                            printData(Location + (UInt32)pos, subBuff);
                            pos += (Int32)(frameSize);
                            break;
                    }
                }
            }
        }

        private UInt32 ID3Size4Calc(byte[] data)
        {
            // ID3タグ用4バイト28ビット(32ビット違うよ)の算出
            return (UInt32)((data[0] << 23) | (data[1] << 15) |
(data[2] << 7) | data[3]);
        }
        private UInt32 Size4Calc(byte[] data)
        {
            return (UInt32)((data[3] << 24) | (data[2] << 16) |
(data[1] << 8) | data[0]);
        }
        private UInt16 Size2Calc(byte[] data)
        {
            return (UInt16)((data[1] << 8) | data[0]);
        }
        private string[] makeHexString(UInt32 Location, byte[] data,
int width)
        {
            List<string> line = new List<string>();
```

```
            StringBuilder sb = new StringBuilder();

            for (int pos = 0; pos < data.Length; pos += width)
            {
                sb.Clear();
                sb.Append(string.Format("{0:X08}", Location + pos));

                for (int idx = 0; idx < width; idx++)
                {
                    if ((pos + idx) == data.Length) break;
                    sb.Append(string.Format(" {0:X02}", data[pos +
idx]));
                }

                line.Add(sb.ToString());
            }

            return line.ToArray();
        }
        private void printData(UInt32 Location, byte[] data, string
caption, object obj)
        {
            string[] line = makeHexString(Location, data, 8);
            for (int idx = 0; idx < line.Length; idx++)
            {
                if (idx != 0)
                {
                    WriteLine(line[idx]);
                }
                else
                {
                    StringBuilder sb = new StringBuilder();
                    sb.Append(line[idx]);
                    sb.Append(' ', (8 + (3 * 8)) - line[idx].Length);

                    WriteLine("{0}  > {1} : \"{2}\"", sb.ToString(),
caption, obj);
                }
                if ((idx % 50) == 0) FlushWriteLine();
            }
            FlushWriteLine();
        }
        private void printData(UInt32 Location, byte[] data, string
caption)
        {
            string[] line = makeHexString(Location, data, 8);
            for (int idx = 0; idx < line.Length; idx++)
            {
                if (idx != 0)
                {
                    WriteLine(line[idx]);
```

```
                }
                else
                {
                    StringBuilder sb = new StringBuilder();
                    sb.Append(line[idx]);
                    sb.Append(' ', (8 + (3 * 8)) - line[idx].Length);

                    WriteLine("{0}  > {1}", sb.ToString(), caption);
                }
                if ((idx % 50) == 0) FlushWriteLine();
            }
            FlushWriteLine();
        }
        private void printData(UInt32 Location, byte[] data)
        {
            string[] line = makeHexString(Location, data, 16);
            for (int idx = 0; idx < line.Length; idx++)
            {
                WriteLine(line[idx]);
                if ((idx % 50) == 0) FlushWriteLine();
            }
            FlushWriteLine();
        }
        public void WriteLine(string str)
        {
            sw.WriteLine(str);
        }
        public void WriteLine(string fmt, params object[] args)
        {
            sw.WriteLine(fmt, args);
        }
    }
}
```

技術資料, wavファイル, RIFF