嘘吐きなzipファイルを作成するコマンド

2015年12月30日
個人的に作っていたコマンドのプロトタイプソースが発掘されたので、適当に手を入れて公開。

LIARZIP.EXEダウンロード

.NET Framework 4.5.2環境下でコンパイルしたもの

LIARZIP.EXEの簡単な説明

ZIPファイル内格納ファイルを隠蔽する方法の検証のために作りました。
※仕事でこれを応用したプログラムを作成する予定でしたが別手段が見つかりボツになりました。

```
H:\Projects\liazip\bin\Release>liarzip
"Liars" zip archiver

usage:
  liarzip    infile outfile : Create an archive "outfile" from "infile"
  liarzip -d infile outfile : Convert to "outfile" from archive "infile"

H:\Projects\liazip\bin\Release>
```

オプション□□d□をつけないで実行した場合は、ZIPファイル作成を行います□infileにZIP化するデータファイル名、outfileにZIPファイル名を指定します。
例えばコマンドライン

```
H:\Projects\liazip\bin\Release>liarzip android.jpg Android.zip
```

を実行すると、ファイル Android.jpg の内容を圧縮して Android.zip を作成します□outfileは常に上書きされます。またinfileは1つのファイルのみ指定できます。ワイルドカードや複数ファイルの指定ができません。

オプションをつけて実行した場合は、ZIPファイルからファイルを取り出します□infile はliarzipで作成したZIPファイル名、outfileにZIPファイルから取り出したデータの保存ファイル名を指定します。
例えばコマンドライン

```
H:\Projects\liazip\bin\Release>liarzip -d Android.zip Android.jpg
```

を実行すると、ファイル Android.zip からデータを取り出して Android.jpg を作成します□outfileは常に上書きされます。

作成されるZIPファイル

上記例で作成された Android.zip は一見普通のZIPファイルです□Windowsでも圧縮フォルダとして中を見ることができます。
…ですが、圧縮したはずの android.jpg は圧縮フォルダにはなく、追加した覚えのない Readme.txt と android.jpg.pkey の2ファイルが見えます。ここでは7-ZIPのCLIコマンド"7z.exe"でAndroid.zipを操作してみます。

```
H:\Projects\liazip\bin\Release>"C:\Program Files\7-Zip\7z.exe" L Android.zip
← 7-zipコマンドで Android.zip をのぞいてみる。
7-Zip [64] 9.20  Copyright (c) 1999-2010 Igor Pavlov  2010-11-18

Listing archive: android.zip

--
Path = android.zip
Type = zip
Physical Size = 1474244

   Date      Time    Attr         Size   Compressed  Name
------------------- ----- ------------ ------------  ------------------------
2015-12-30 04:06:46 ....A           76           76  Readme.txt
← 追加した覚えのないReadme.txtとandroid.jpg.pkeyがZIPファイルに格納されている。
2015-12-30 04:02:20 ....A          884          884  android.jpg.pkey
------------------- ----- ------------ ------------  ------------------------
                                   960          960  2 files, 0 folders

H:\Projects\liazip\bin\Release>"C:\Program Files\7-Zip\7z.exe" X Android.zip
← Android.zipを解凍してみる。

7-Zip [64] 9.20  Copyright (c) 1999-2010 Igor Pavlov  2010-11-18

Processing archive: android.zip

Extracting  Readme.txt
Extracting  android.jpg.pkey

Everything is Ok

Files: 2
Size:       960
Compressed: 1474244

H:\Projects\liazip\bin\Release>dir android*.*
 ドライブ H のボリューム ラベルは HCER です
 ボリューム シリアル番号は FE01-21D0 です

 H:\Projects\liazip\bin\Release のディレクトリ
```

```
2015/12/30  04:02               884 android.jpg.pkey  ← android.jpg がない！。
2015/12/30  04:06         1,474,244 android.zip
      2 個のファイル      1,475,128 バイト
      0 個のディレクトリ  1,731,419,164,672 バイトの空き領域

H:\Projects\liazip\bin\Release>type Readme.txt
This archive is PACX format. Attached the "pkey" crypt data for android.jpg.
  ← Readme.txtの記載にある PACX フォーマットってなんだ？
H:\Projects\liazip\bin\Release>
```

何故か追加されたReadme.txtにはPACXフォーマットだとか暗号化データpkeyだとか書かれています。

……全部嘘っぱちです。😬

Android.zipには android.jpg が確かに格納されていますが、標準の方法では参照ができない状態になっています。
ファイル android.jpg.pkey や Readme.txt は仕掛けがわからない人を欺くための撒き餌です。こっちに食い付いてくれればしばらくからくりに気が付くことはないでしょう。
※ZIPファイルサイズに対して android.jpg.pkey や Readme.txt がやけに小さい事に気が付けば、もしかしたら「こわれたZIPファイルかな？」と思うかもしれません。

今度はオプション -d を使ってAndroid.zip を解凍します。

```
H:\Projects\liazip\bin\Release>liarzip -d Android.zip android.jpg

H:\Projects\liazip\bin\Release>dir android*.*
 ドライブ H のボリューム ラベルは HCER です
 ボリューム シリアル番号は FE01-21D0 です

 H:\Projects\liazip\bin\Release> のディレクトリ

2015/12/30  05:15         1,473,364 android.jpg
2015/12/30  04:02               884 android.jpg.pkey
2015/12/30  04:06         1,474,244 android.zip
      3 個のファイル      2,948,492 バイト
      0 個のディレクトリ  1,731,422,781,440 バイトの空き領域

H:\Projects\liazip\bin\Release>
```

無事 android.jpg を取り出すことができました。もちろん android.jpg はJPEG画像として使用することができます。

何に使うのこれ？

何に使えるんでしょうね。わかりません。

元々は設定ミス等で誤った宛先にメール送信してしまった際に添付されたZIPファイルの参照をできるだけ防ぐ方法の一つとして検討されていました。
※ZIPのパスワードはすぐ解除できてしまいます。パスワードをかけたZIPファイルそのものを LIARZIP.EXE でアーカイブし直せば(メール添付の際に自動で処理します)、普通の人には壊れたZIPにしか見えずなかなかカラクリがわからんだろう、と。

でも結局、目的の仕掛けのプラグインでPGPを間接的にですが利用出来る事がわかったのでそちらを採用しました。

ソースコード

System.IO.Compression ネームスペースにZIPやGZIP形式のアーカイブを操作するいろいろなクラスがそろっています。このソースでは圧縮解凍にgzipのストリーム操作で利用しています。
ZIPアーカイブファイルそのものの作成には使っていません。自前でZIPアーカイブを組み立てています。

生成されるZIPアーカイブ自体は不正のない形式です□PK0506ヘッダのPK0201ヘッダを指すポインタに操作を加え、隠蔽対象のファイルエントリが見えないようにしています。

liarzip.cs

```
using System;
using System.Text;
using System.IO;
using System.IO.Compression;

namespace liarzip
{
    class liarzip
    {
        static void Main(string[] args)
        {
            zipUtil ac = new zipUtil();

            if ((2 > args.Length) || (3 < args.Length))
            {
                Console.WriteLine("\"Liars\" zip archiver\n");
                Console.WriteLine("usage:");
                Console.WriteLine("  liarzip    infile outfile : Create
an archive \"outfile\" from \"infile\"");
                Console.WriteLine("  liarzip -d infile outfile :
Convert to \"outfile\" from archive \"infile\"");
                return;
            }
            else if (2 == args.Length)
            {
                // zip
                ac.compressFile(args[0], args[1]);
            }
            else
            {
                // unzip
                ac.uncompressFile(args[1], args[2]);

            }

            return;
```

```
        }
    }

    class zipUtil
    {
        private class writeInfo
        {
            public UInt32 crc32;
            public UInt32 size;
            public UInt32 compressSize;
            public String fileName;
            public DateTime timeStamp;
            public UInt32 writeSize;
        }
        private class fileResult
        {
            public UInt32 crc32;
            public UInt32 fileSize;
            public DateTime dateTime;
        }

        //参照 http://ja.wikipedia.org/wiki/巡回冗長検査
        private class CRCUtil
        {
            private static UInt32[] CRCTable = new UInt32[256];

            static CRCUtil()
            {
                for (UInt32 idx = 0; idx < 256; idx++)
                {
                    UInt32 bc = idx;
                    for (int j = 0; j < 8; j++)
                    {
                        bc = (bc & 1) != 0 ? (0xEDB88320 ^ (bc >> 1)) :
(bc >> 1);
                    }
                    CRCTable[idx] = bc;
                }
            }
            public UInt32 crc32FromText(String textString)
            {
                UInt32 ans = 0xFFFFFFFF;

                byte[] buff =
Encoding.UTF8.GetBytes(textString.ToCharArray());

                for (long bidx = 0; bidx < buff.Length; bidx++)
                {
                    ans = CRCTable[(ans ^ buff[bidx]) & 0x000000FF] ^
(ans >> 8);
                }
```

```
                return ans ^ 0xFFFFFFFF;
            }
            public UInt32 crc32FromByteArray(byte[] buff)
            {
                return crc32FromByteArray(buff, buff.Length);
            }
            public UInt32 crc32FromByteArray(byte[] buff, Int32 len)
            {
                UInt32 ans = 0xFFFFFFFF;

                for (long bidx = 0; bidx < len; bidx++)
                {
                    ans = CRCTable[(ans ^ buff[bidx]) & 0x000000FF] ^
(ans >> 8);
                }

                return ans ^ 0xFFFFFFFF;
            }
            public fileResult crc32FromFile(String filePath)
            {
                fileResult ans = new fileResult();

                UInt32 crc = 0xFFFFFFFF;
                Int32 buffSize = 16 * 1024;
                byte[] buff = new byte[buffSize];

                FileInfo fi = new FileInfo(filePath);
                ans.fileSize = (UInt32)fi.Length;
                ans.crc32 = crc;
                ans.dateTime = fi.CreationTime;

                using (FileStream bstream = fi.OpenRead())
                {
                    Int32 readBytes = bstream.Read(buff, 0, buffSize);

                    for (Int32 bidx = 0; bidx < readBytes; bidx++)
                    {
                        crc = CRCTable[(crc ^ buff[bidx]) & 0x000000FF]
^ (crc >> 8);
                    }
                }
                ans.crc32 = crc ^ 0xFFFFFFFF;

                return ans;
            }
        }

        private CRCUtil cu = new CRCUtil();

        public void uncompressFile(String inFilePath, String
```

```
outFilePath)
        {
            String tempName = Path.GetRandomFileName();

            FileInfo fakezipinfo = new FileInfo(inFilePath);
            FileStream fakezipfs = fakezipinfo.OpenRead();
            FileStream tempfs = new FileStream(tempName,
FileMode.Create);

            long skipSize = seekToInfoBody(ref fakezipfs,
fakezipinfo.Length);    //擬装zip内を圧縮されたターゲット本体まで読み飛ばす

            fakezipfs.Seek(skipSize + 2, SeekOrigin.Begin);
            tempfs.WriteByte(0x1f);   // gzip シグネチャ復元
            tempfs.WriteByte(0x8b);   // gzip シグネチャ復元
            fakezipfs.CopyTo(tempfs);
            fakezipfs.Close();
            tempfs.Close(); //圧縮ターゲット取り出し完了


            //圧縮されたターゲットの復元

            tempfs = new FileStream(tempName, FileMode.Open,
FileAccess.Read);

            FileStream target = new FileStream(outFilePath,
FileMode.Create);
            GZipStream gzd = new GZipStream(tempfs,
CompressionMode.Decompress, true);

            gzd.CopyTo(target);
            gzd.Close();
            target.Close();
            tempfs.Close();

            //圧縮されたターゲットの削除
            FileInfo zi = new FileInfo(tempName);
            zi.Delete();

        }
        public void compressFile(String inFilePath, String outFilePath)
        {
            //ターゲットファイルをgzip圧縮する□zipではなくgzipです。
            FileInfo targetFi = new FileInfo(inFilePath);            //日
付保持用
            FileInfo gzipFi = compressToGzipFile(inFilePath);        //圧
縮結果はtemp名

            //作成する擬装ZIPファイルのファイルストリーム
            FileStream wz = new FileStream(outFilePath,
FileMode.Create);
```

```
            // Readme.txtのPK0304ヘッダとデータを出力。
            writeInfo readme0304 = writeTextPK0304(ref wz,
"Readme.txt", "This archive is PACX format. Attached the \"pkey\" crypt
data for " + inFilePath + ".");

            //偽ターゲットファイルのPK0304ヘッダとデータを出力。
            writeInfo fake0304 = writeDummyPK0304(ref wz, inFilePath);

            //圧縮されたターゲット本体のPK0304ヘッダとデータを出力。
            writeInfo target0304 = writeTargetPK0304(ref wz, gzipFi,
inFilePath);

            //圧縮されたターゲットのPK0102ヘッダを出力。このヘッダはPK0506からポ
イントされない。
            long target0102 = outputZipEntryHeaderPK0102(ref wz,
target0304.crc32, target0304.size, target0304.compressSize,
target0304.fileName, (UInt32)(readme0304.writeSize +
fake0304.writeSize), calcDate(targetFi.CreationTime),
calcTime(targetFi.CreationTime));

            // Readme.txtのPK0102ヘッダを出力。
            long readme0102 = outputZipEntryHeaderPK0102(ref wz,
readme0304.crc32, readme0304.size, readme0304.compressSize,
readme0304.fileName, (UInt32)( 0 ), calcDate(readme0304.timeStamp),
calcTime(readme0304.timeStamp));

            //偽ターゲットファイルのPK0102ヘッダを出力。
            long fake0102 = outputZipEntryHeaderPK0102(ref wz,
fake0304.crc32, fake0304.size, fake0304.compressSize,
fake0304.fileName, (UInt32)(readme0304.writeSize),
calcDate(fake0304.timeStamp), calcTime(fake0304.timeStamp));

            // PK0102ヘッダの開始位置をポイントするPK0506ヘッダを出力。ファイル
が二つと報告させる。
            long finish0506 = outputZipEntryHeaderPK0506(ref wz,
(UInt32)( readme0102 + fake0102 ), (UInt32)(readme0304.writeSize +
fake0304.writeSize + target0304.writeSize + target0102), 2);


            wz.Close();
            gzipFi.Delete();
        }

        private FileInfo compressToGzipFile(String inFilePath)
        {
            String tempName = Path.GetRandomFileName();
            DateTime dt = DateTime.Now;

            FileInfo targetfs = new FileInfo(inFilePath);
```

```
            FileStream tempfs = new FileStream(tempName,
FileMode.Create, FileAccess.ReadWrite);
            GZipStream gzc = new GZipStream(tempfs,
CompressionMode.Compress, true);

            FileStream r = targetfs.OpenRead();

            r.CopyTo(gzc);
            r.Close();
            gzc.Close();
            tempfs.Flush();
            tempfs.Seek(0, SeekOrigin.Begin);
            tempfs.WriteByte(0xfa);   // gzipシグネチャを改変
            tempfs.WriteByte(0xce);   // gzipシグネチャを改変
            tempfs.Close();

            return new FileInfo(tempName);
        }
        private writeInfo writeTextPK0304(ref FileStream wf, String
dummyName, String dummyText)
        {
            writeInfo ans = new writeInfo();

            //byte[] dummyTextAry =
Encoding.UTF8.GetBytes(dummyText.ToCharArray());
            byte[] dummyTextAry =
Encoding.UTF8.GetBytes(dummyText.ToCharArray());

            ans.crc32 = cu.crc32FromText(dummyText);
            ans.size = (UInt32)dummyTextAry.Length;
            ans.compressSize = ans.size;
            ans.fileName = dummyName;
            ans.timeStamp = DateTime.Now;

            //ヘッダ書き出し
            ans.writeSize = (UInt32)outputZipEntryHeaderPK0304(ref wf,
ans.crc32, ans.size, ans.compressSize, ans.fileName,
calcDate(ans.timeStamp), calcTime(ans.timeStamp));

            //テキスト書き出し
            wf.Write(dummyTextAry, 0, dummyTextAry.Length);
            ans.writeSize += (UInt32)dummyTextAry.Length;

            return ans;
        }
        private writeInfo writeDummyPK0304(ref FileStream wf, String
inFilePath)
        {
            writeInfo ans = new writeInfo();
            fileResult crc = cu.crc32FromFile(inFilePath);
```

```
            Random rng = new Random();
            byte[] df = new byte[rng.Next(1020) + 8];

            rng.NextBytes(df);

            // シグネチャっぽい文字を先頭に埋めておく
            df[0] = 0x50; // P
            df[1] = 0x61; // a
            df[2] = 0x63; // c
            df[3] = 0x58; // X

            df[4] = 0x30; // 0
            df[5] = 0x31; // 1
            df[6] = 0x30; // 0
            df[7] = 0x30; // 0

            UInt32 bfcrc = cu.crc32FromByteArray(df); // ランダムに埋めた配列のCRC32計算

            ans.crc32 = bfcrc;
            ans.size = (UInt32)df.Length;
            ans.compressSize = (UInt32)df.Length;
            ans.fileName = inFilePath + ".pkey";
            ans.timeStamp = crc.dateTime;

            // ヘッダ書き出し
            ans.writeSize = (UInt32)outputZipEntryHeaderPK0304(ref wf,
ans.crc32, ans.size, ans.compressSize, ans.fileName,
calcDate(ans.timeStamp), calcTime(ans.timeStamp));

            //配列データ書き出し
            wf.Write(df, 0, df.Length);
            ans.writeSize += (UInt32)df.Length;

            return ans;
        }
        private writeInfo writeTargetPK0304(ref FileStream wf, FileInfo
gzfi, String inFilePath)
        {
            writeInfo ans = new writeInfo();
            fileResult crc = cu.crc32FromFile(gzfi.Name);
            FileStream targetfs = gzfi.OpenRead();

            ans.crc32 = crc.crc32;
            ans.size = (UInt32)crc.fileSize;
            ans.compressSize = (UInt32)crc.fileSize;
            ans.fileName = inFilePath + ".pacx";
            ans.timeStamp = crc.dateTime;

            // ヘッダ書き出し
            ans.writeSize = (UInt32)outputZipEntryHeaderPK0304(ref wf,
```

```
ans.crc32, ans.size, ans.compressSize, ans.fileName,
calcDate(ans.timeStamp), calcTime(ans.timeStamp));

            //圧縮ターゲット書き出し
            targetfs.CopyTo(wf);
            targetfs.Close();
            ans.writeSize += ans.size;

            return ans;
        }
        private long outputZipEntryHeaderPK0102(ref FileStream wf,
UInt32 crc32, UInt32 dataByteLength, UInt32 dataCompLength, String
fileName, UInt32 positionPK34, UInt16 date, UInt16 time)
        {
            byte[] nameByteAry =
Encoding.GetEncoding("shift_jis").GetBytes(fileName.ToCharArray());
            EncodingInfo[] f = Encoding.GetEncodings();

            byte[] header = new byte[]{
                                 0x50, 0x4b, 0x01, 0x02
// P K 0x01 0x02  *signature
                                ,0x14, 0x00
// format version
                                ,0x14, 0x00
// minimum version
                                ,0x00, 0x00
// option
                                ,0x00, 0x00
// no compress

                                ,(byte)( time & 0x00FF)
// time +0
                                ,(byte)((time & 0xFF00) >>8  )
// time +1
                                ,(byte)( date & 0x00FF)
// date +0
                                ,(byte)((date & 0xFF00) >>8  )
// date +1

                                ,(byte)( crc32 & 0x000000FF      )
// CRC32 +0
                                ,(byte)((crc32 & 0x0000FF00) >>8 )
// CRC32 +1
                                ,(byte)((crc32 & 0x00FF0000) >>16)
// CRC32 +2
                                ,(byte)((crc32 & 0xFF000000) >>24)
// CRC32 +3

                                ,(byte)( dataCompLength & 0x000000FF
)     // compress size +0
                                ,(byte)((dataCompLength & 0x0000FF00)
```

```
>>8 )     // compress size +1
                                    ,(byte)((dataCompLength & 0x00FF0000)
>>16)     // compress size +2
                                    ,(byte)((dataCompLength & 0xFF000000)
>>24)     // compress size +3

                                    ,(byte)( dataByteLength & 0x000000FF
)     // uncompress size +0
                                    ,(byte)((dataByteLength & 0x0000FF00)
>>8 )     // uncompress size +1
                                    ,(byte)((dataByteLength & 0x00FF0000)
>>16)     // uncompress size +2
                                    ,(byte)((dataByteLength & 0xFF000000)
>>24)     // uncompress size +3

                                    ,(byte)( nameByteAry.Length &
0x000000FF     )  // dummy filename size +0
                                    ,(byte)((nameByteAry.Length &
0x0000FF00) >>8)  // dummy filename size +1

                                    ,0x00, 0x00
// extra data size
                                    ,0x00, 0x00
// comment length
                                    ,0x00, 0x00
// disknum
                                    ,0x00, 0x00
// in attr
                                    ,0x20, 0x00, 0x00, 0x00
// orign attr *Windows

                                    ,(byte)( positionPK34 & 0x000000FF
)       // PK34 Enery Position +0
                                    ,(byte)((positionPK34 & 0x0000FF00) >>8
)       // PK34 Enery Position +1
                                    ,(byte)((positionPK34 & 0x00FF0000)
>>16)       // PK34 Enery Position +2
                                    ,(byte)((positionPK34 & 0xFF000000)
>>24)       // PK34 Enery Position +3
                           }; // total 46 byte

            wf.Write(header, 0, header.Length);
            wf.Write(nameByteAry, 0, nameByteAry.Length);

            return header.Length + nameByteAry.Length;
        }
        private long outputZipEntryHeaderPK0506(ref FileStream wf,
UInt32 sizePK12, UInt32 positionPK12, UInt16 fileCount)
        {
            byte[] header = new byte[]{
                                     0x50, 0x4b, 0x05, 0x06
```

```
// P K 0x05 0x06   *signature
                                ,0x00, 0x00
// disknum              * 0 only
                                ,0x00, 0x00
// start disknum        * 0 only

                                ,(byte)( fileCount & 0x00FF)
// disk directory entry +0
                                ,(byte)((fileCount & 0xFF00) >>8  )
// disk directory entry +1
                                ,(byte)( fileCount & 0x00FF)
// directory entry +0
                                ,(byte)((fileCount & 0xFF00) >>8  )
// directory entry +1

                                ,(byte)( sizePK12 & 0x000000FF
)      // PK12 Entry size +0
                                ,(byte)((sizePK12 & 0x0000FF00) >>8
)      // PK12 Entry size +1
                                ,(byte)((sizePK12 & 0x00FF0000) >>16
)      // PK12 Entry size +2
                                ,(byte)((sizePK12 & 0xFF000000) >>24
)      // PK12 Entry size +3

                                ,(byte)( positionPK12 & 0x000000FF
)      // PK12 First Entry position +0
                                ,(byte)((positionPK12 & 0x0000FF00) >>8
)      // PK12 First Entry position +1
                                ,(byte)((positionPK12 & 0x00FF0000)
>>16 )      // PK12 First Entry position +2
                                ,(byte)((positionPK12 & 0xFF000000)
>>24 )      // PK12 First Entry position +3
                                ,0x00, 0x00
// comment length
                    }; // total 22 byte

            wf.Write(header, 0, header.Length);

            return header.Length;
        }
        private long outputZipEntryHeaderPK0304(ref FileStream wf,
UInt32 crc32, UInt32 dataByteLength, UInt32 dataCompLength, String
fileName, UInt16 date, UInt16 time)
        {
            byte[] nameByteAry =
Encoding.GetEncoding("shift_jis").GetBytes(fileName.ToCharArray());
            byte[] header = new byte[]{
                                 0x50, 0x4b, 0x03, 0x04
// P K 0x03 0x04  *signature
                                ,0x14, 0x00
// minimum version
```

```
                                    ,0x00, 0x00
// option
                                    ,0x00, 0x00
// no compress

                                    ,(byte)( time & 0x00FF)
// time +0
                                    ,(byte)((time & 0xFF00) >>8  )
// time +1
                                    ,(byte)( date & 0x00FF)
// date +0
                                    ,(byte)((date & 0xFF00) >>8  )
// date +1

                                    ,(byte)( crc32 & 0x000000FF      )
// CRC32 +0
                                    ,(byte)((crc32 & 0x0000FF00) >>8 )
// CRC32 +1
                                    ,(byte)((crc32 & 0x00FF0000) >>16)
// CRC32 +2
                                    ,(byte)((crc32 & 0xFF000000) >>24)
// CRC32 +3

                                    ,(byte)( dataCompLength & 0x000000FF
)          // compress size +0
                                    ,(byte)((dataCompLength & 0x0000FF00)
>>8 )          // compress size +1
                                    ,(byte)((dataCompLength & 0x00FF0000)
>>16)          // compress size +2
                                    ,(byte)((dataCompLength & 0xFF000000)
>>24)          // compress size +3

                                    ,(byte)( dataByteLength & 0x000000FF
)          // uncompress size +0
                                    ,(byte)((dataByteLength & 0x0000FF00)
>>8 )          // uncompress size +1
                                    ,(byte)((dataByteLength & 0x00FF0000)
>>16)          // uncompress size +2
                                    ,(byte)((dataByteLength & 0xFF000000)
>>24)          // uncompress size +3

                                    ,(byte)( nameByteAry.Length &
0x000000FF     )      // dummy filename size +0
                                    ,(byte)((nameByteAry.Length &
0x0000FF00) >>8)      // dummy filename size +1
                                    ,0x00, 0x00
// extra data size
            }; // total 30 byte

            wf.Write(header, 0, header.Length);
            wf.Write(nameByteAry, 0, nameByteAry.Length);
```

```
            return header.Length + nameByteAry.Length;
        }
        private long seekToInfoBody(ref FileStream rfs, long rfsize)
        {

            const Int32 PKSIGLEN = 4;
            const Int32 PK0304LEN = 30;
            const Int32 PK0102LEN = 46;
            const Int32 PK0506LEN = 22;

            UInt16 filenameLength;
            UInt32 bodyLength;
            UInt16 extraBufLength;
            UInt16 commentStrLength;

            byte[] pkxxxxheader = new byte[PKSIGLEN];
            byte[] pk0304header = new byte[PK0304LEN - PKSIGLEN];
            byte[] pk0102header = new byte[PK0102LEN - PKSIGLEN];
            byte[] pk0506header = new byte[PK0506LEN - PKSIGLEN];

            bool breakFlag = false;
            long ans = 0;
            long addSeekSize = 0;
            long pk0304count = 0;

            String signature = "PKxxxx";

            //擬装ZIPアーカイブのヘッダスキップ
            while (false == breakFlag)
            {
                if (ans > rfsize) break;

                addSeekSize = 0;
                rfs.Read(pkxxxxheader, 0, PKSIGLEN);

                signature = Convert.ToChar(pkxxxxheader[0]).ToString()
+ Convert.ToChar(pkxxxxheader[1]).ToString() +
String.Format("{0:X02}{1:X02}", pkxxxxheader[2], pkxxxxheader[3]);

                switch (signature)
                {
                    case "PK0304":
                        pk0304count++;
                        rfs.Read(pk0304header, 0, PK0304LEN -
PKSIGLEN);
                        filenameLength = (UInt16)(((0xFFFF &
pk0304header[27 - PKSIGLEN])     <<  8 ) | pk0304header[26 -
PKSIGLEN]);
                        bodyLength     = (UInt32)(((0xFFFFFFFF &
pk0304header[21 - PKSIGLEN]) << 24 ) | ((0xFFFFFFFF & pk0304header[20 -
```

```
PKSIGLEN]) << 16)
| ((0xFFFFFFFF & pk0304header[19 - PKSIGLEN]) <<  8)
| pk0304header[18 - PKSIGLEN]);
                        extraBufLength = (UInt16)(((0xFFFF &
pk0304header[29 - PKSIGLEN])     <<  8 ) | pk0304header[28 -
PKSIGLEN]);

                        //3回目に現れる PK0304ヘッダが目的の圧縮されたターゲット
のもの
                        if (3 != pk0304count)
                        {
                            addSeekSize = (long)(filenameLength +
bodyLength + extraBufLength);
                        }
                        else
                        {
                            addSeekSize = (long)(filenameLength +
extraBufLength);
                            breakFlag = true;
                        }
                        ans += PK0304LEN + addSeekSize;
                        break;

                    case "PK0102":
                        rfs.Read(pk0102header, 0, PK0102LEN -
PKSIGLEN);
                        filenameLength   = (UInt16)(((0xFFFF &
pk0102header[29 - PKSIGLEN]) << 8) | pk0102header[28 - PKSIGLEN]);
                        extraBufLength   = (UInt16)(((0xFFFF &
pk0102header[31 - PKSIGLEN]) << 8) | pk0102header[30 - PKSIGLEN]);
                        commentStrLength = (UInt16)(((0xFFFF &
pk0102header[33 - PKSIGLEN]) << 8) | pk0102header[32 - PKSIGLEN]);

                        addSeekSize = (long)(filenameLength +
extraBufLength + commentStrLength);
                        ans += PK0102LEN + addSeekSize;
                        break;

                    case "PK0506":
                        rfs.Read(pk0506header, 0, PK0506LEN -
PKSIGLEN);
                        commentStrLength = (UInt16)(((0xFFFF &
pk0506header[21 - PKSIGLEN]) << 8) | pk0506header[20 - PKSIGLEN]);

                        addSeekSize = (long)(commentStrLength);
                        ans += PK0506LEN + addSeekSize;
                        breakFlag = true; //現行だとこの後ろに続くことはない
ので、終わらせておく
                        break;

                    default:
```

```
                        addSeekSize = 0;
                        breakFlag = true;
                        break;
                    }

                    rfs.Seek(addSeekSize, SeekOrigin.Current);

                }

                return ans;
            }
            private UInt16 calcDate(DateTime dt)
            {
                return (UInt16)((0xFFFF & ((dt.Year - 1980) << 9)) |
(0xFFFF & (dt.Month << 5)) | dt.Day);
            }
            private UInt16 calcTime(DateTime dt)
            {
                return (UInt16)((0xFFFF & (dt.Hour << 11)) | (0xFFFF &
(dt.Minute << 5)) | (0xFFFF & (dt.Second >> 1)));
            }
        }
}
```

技術資料, zip, C#, windows