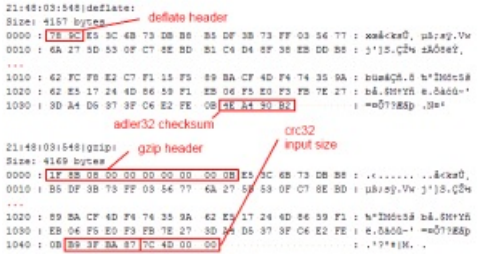
GZIP bzw. deflate Streams mit Free Pascal

Komprimierte Datenübertragung

GZip und deflate(ZLib) unterscheidet sich lediglich durch den Header und die Checksumme, denn beide benutzt im Eigentlichen die selbe Methode zum Komprimieren (deflate = Luft heraus lassen :))



(/media/gzip.png)

Unterschiede Deflate / GZip

Unter Free Pascal steht das Package paszlib zur Verfügung, welches die benötigten Funktionen zum Behandeln von deflated Blöcken zur Verfügung stellt. Mit Hilfe dieser Funktionen lassen sich alle benötigten Funktionen deflate (raw), GZip, ZLib, inflate (raw) und ZUncompressStream erzeugen. ZUncompessStream übernimmt die Identifizierung des Typs und nutzt die Funktion inflate zum entpacken des deflated Teils.

Die Routinen sollten dazu in der Lage sein sowohl deflated (ZLib) und gziped Streams zum Versand zu erzeugen, als auch Empfangene Streams dieser Formate wieder zu entpacken!

Aufbau

Deflate (ZLib): ZLIB Compressed Data Format Specification (https://www.rfc-editor.org/rfc/rfc1950)

- 2 bytes Defines the compression mode
  - 1 byte $78 CMF Compression Method and flags
    "deflate" compression method with a window size up to 32K
  - 1 byte FLG
    - bit 0-4 FCHECK (check bits for CMF and FLG)
    - bit 5 FDICT (preset dictionary)
    - bit 6-7 FLEVEL (compression level)

    Examples: (List of file signatures (https://en.wikipedia.org/wiki/List_of_file_signatures)
)
    - $01 No Compression (no preset dictionary)
    - $5E Best speed (no preset dictionary)
    - $9C Default Compression (no preset dictionary)
    - $DA Best Compression (no preset dictionary)
    - $20 No Compression (with preset dictionary)
    - $7D Best speed (with preset dictionary)
    - $BB Default Compression (with preset dictionary)
    - $F9 Best Compression (with preset dictionary)
- 4 bytes DICTID Present only when FLG.FDICT is set.
- deflated stream
- 4 bytes adler32 checksum

GZip: GZIP file format specification (https://www.ietf.org/rfc/rfc1952.txt)

- 2 bytes $1f $8b (IDentification)
- 1 byte $08 Compression Method = deflate
- 1 byte $00 FLaGs
  - bit 0   FTEXT - indicates file is ASCII text (can be safely ignored)
  - bit 1   FHCRC - there is a CRC16 for the header immediately following the header
  - bit 2   FEXTRA - extra fields are present
  - bit 3   FNAME - the zero-terminated filename is present. encoding; ISO-8859-1.
  - bit 4   FCOMMENT - a zero-terminated file comment is present. encoding: ISO-8859-1
  - bit 5-7   reserved
- 4 bytes $00000000 Modification TIME = no time stamp is available
- 1 byte $00 eXtra FLags
  - 00 - default compression
  - 02 - compressor used maximum compression, slowest algorithm
  - 04 - compressor used fastest algorithm
- 1 byte $0b Operating System = NTFS filesystem (NT) for textfiles (line ending)
  - 00 - FAT filesystem (MS-DOS, OS/2, NT/Win32)
  - 01 - Amiga 02 - VMS (or OpenVMS)
  - 03 - Unix
  - 04 - VM/CMS
  - 05 - Atari TOS
  - 06 - HPFS filesystem (OS/2, NT)
  - 07 - Macintosh
  - 08 - Z-System
  - 09 - CP/M
  - 0A - TOPS-20
  - 0B - NTFS filesystem (NT)
  - 0C - QDOS
  - 0D - Acorn RISCOS
  - FF - unknown
- deflated stream
- crc32 checksum
- input size

GZIPUtils.pas (/code/gziputils.pas?mode=download) Pascal (9,94 kByte) 16.04.2022 09:02

```
//
////////////////////////////////////////////////////////////////////////////////

//  Title.............. :  GZIP / deflate / inflate Streams with PasZLib
```

```
//
//  Modulname ......... :  giputils.pas
//  Type .............. :  Unit
//  Author ............ :  Udo Schmal
//  Development Status  :  20.03.2016
//  Operating System .. :  Win32/Win64
//  IDE ............... :  Delphi & Lazarus
//
*******************************************************************************

unit GZIPUtils;
{$ifdef fpc}
  {$mode objfpc}
{$endif}
{$H+}

interface

uses Classes, SysUtils, PasZLib, zbase;

type
  TZCompressionLevel = (zcNone, zcFastest, zcDefault, zcMax);

  TZStreamType = (
    zsZLib, // standard zlib stream (deflate header)
    zsGZip, // gzip stream (with gzip header)
    zsRaw,  // raw stream (without any header)
    zsNo    // no compression
  );

  TGZipHeaderFLags = (
    FTEXT,      // bit 0 - indicates file is ASCII text (can be safely
ignored)
    FHCRC,      // bit 1 - there is a CRC16 for the header immediately
following the header
    FEXTRA,     // bit 2 - extra field is present
    FNAME,      // bit 3 - the zero-terminated filename is present. encoding;
ISO-8859-1.
    FCOMMENT    // bit 4 - a zero-terminated file comment is present.
encoding: ISO-8859-1
  );
  TFlags = set of TGZipHeaderFLags;

const
  Z_BUFSIZE = 65536; // 16384, 32768, 65536;
  GZIP_WBITS = MAX_WBITS + 16; // GZip header
  ZLIB_WBITS = MAX_WBITS; // zlib header
  RAW_WBITS = -MAX_WBITS; // deflate raw stream (without any header)

  ZLevels: array[TZCompressionLevel] of Shortint = (
```

```
    Z_NO_COMPRESSION,
    Z_BEST_SPEED,
    Z_DEFAULT_COMPRESSION,
    Z_BEST_COMPRESSION
  );

function zipStream(inStream, outStream: TMemoryStream; level:
TZCompressionLevel = zcDefault; streamType: TZStreamType = zsZLib): boolean;
function unzipStream(inStream, outStream: TMemoryStream): boolean;

implementation

function zipStream(inStream, outStream: TMemoryStream; level:
TZCompressionLevel = zcDefault; streamType: TZStreamType = zsZLib): boolean;
var
  zstream: z_stream;
  crc, size, adler, headerSize: longword;
begin
  result := false;
  inStream.Position := 0; // goto start of input stream
  outStream.Position := 0; // goto start of output stream
  if StreamType = zsGZip then //add GZip Header
  begin
    size := inStream.Size;
    crc := crc32(0, Pointer(inStream.Memory), size);
    outStream.WriteWord($8b1f); //GZip IDentification
    outStream.WriteByte($08); //Compression Method = deflate
    // 00 - store (no compression)
    // 01 - compress
    // 02 - pack
    // 03 - lzh
    // 04..07 - reserved
    // 08 - deflate
    outStream.WriteByte($00); //FLags
    // bit 0   FTEXT - indicates file is ASCII text (can be safely ignored)
    // bit 1   FHCRC - there is a CRC16 for the header immediately following
the header
              // continuation of multi-part gzip file, part number present
    // bit 2   FEXTRA - extra field is present
    // bit 3   FNAME - the zero-terminated filename is present. encoding;
ISO-8859-1.
    // bit 4   FCOMMENT  - a zero-terminated file comment is present.
encoding: ISO-8859-1
    // bit 5 -7   reserved
    outStream.WriteDWord($00000000); //Modification TIME = no time stamp is
available (UNIX time format)
    outStream.WriteByte($00); //eXtra FLags (depend on compression method)
    // 00 - default compression
    // 02 - compressor used maximum compression, slowest algorithm
    // 04 - compressor used fastest algorithm
```

```
    outStream.WriteByte({$ifdef win32}$0b{$else}$03{$endif}); //Operating
System = NTFS filesystem (NT)
    // 00 - FAT filesystem (MS-DOS, OS/2, NT/Win32)
    // 01 - Amiga
    // 02 - VMS (or OpenVMS)
    // 03 - Unix
    // 04 - VM/CMS
    // 05 - Atari TOS
    // 06 - HPFS filesystem (OS/2, NT)
    // 07 - Macintosh
    // 08 - Z-System
    // 09 - CP/M
    // 0A - TOPS-20
    // 0B - NTFS filesystem (NT)
    // 0C - QDOS
    // 0D - Acorn RISCOS
    // FF - unknown
  end
  else if StreamType = zsZLib then //adler32
  begin
    outStream.WriteWord($9c78); //ZLib Header
    adler := adler32(0, Z_NULL, 0);
    adler := adler32(adler, Pointer(inStream.Memory), inStream.Size);
  end;

  // deflate raw stream
  headerSize := outStream.Position;
  zstream.next_in := inStream.Memory;
  zstream.avail_in := inStream.Size;
  outStream.SetSize(headerSize + ((inStream.Size + (inStream.Size div 10) +
12) + 255) and not 255);
  zstream.next_out := outStream.Memory + headerSize;
  zstream.avail_out := outStream.Size - headerSize;
  if deflateInit2(zstream, ZLevels[level], Z_DEFLATED, RAW_WBITS, 8,
Z_DEFAULT_STRATEGY) < Z_OK then Exit;
  deflate(zstream, Z_FINISH);
  result := not (deflateEnd(zstream) < 0);
  outStream.SetSize(zstream.total_out + headerSize);
  outStream.Position := zstream.total_out + headerSize;

  if result and (StreamType = zsGZip) then // add checksum and size
  begin
    outStream.WriteDWord(crc); // CRC32 (CRC-32)
    outStream.WriteDWord(size); // ISIZE (Input SIZE)
  end
  else if result and (StreamType = zsZLib) then // add adler32 checksum
    outStream.WriteDWord(SwapEndian(adler)); // adler32 checksum

  outStream.Position := 0; // goto start of result stream
end;
```

```
function crc16(crc: word; data: Pbyte; len: Cardinal): word;
var sum: Cardinal;
begin
  sum := crc;
  while len > 1 do
  begin
    inc(sum, PWord(data)^);
    inc(data, 2);
    dec(len, 2)
  end;
  // Add left-over byte, if any
  if len > 0 then
    inc(sum, PByte(data)^);
  // Fold 32-bit sum to 16 bits
  while (sum shr 16) > 0 do
    sum := (sum and $ffff) + (sum shr 16);
  result := not word(sum);
end;

function unzipStream(inStream, outStream: TMemoryStream): boolean;
var
  streamType: TZStreamType;
  zstream: z_stream;
  hdr, crc, adler, adler32in, crcGZin, sizeGZin, delta, headerSize,
modificationtime: longword;
  len, crcH, crcHeader: word;
  b: byte;
  flags: TFlags;
//  sFilename, sComment: string;
begin
  result := false;
  inStream.Position := 0; // goto start of input stream
  outStream.Position := 0; // goto start of output stream
  sizeGZin := 0;
  hdr := inStream.ReadDWord;
  if (hdr and $00088B1F) = $00088B1F then // gzip header (deflate method)
  begin
    streamType := zsGZip; // GZIP format
    modificationtime := inStream.ReadDWord; //Modification TIME (UNIX time
format)
    inStream.ReadWord; // eXtra FLags & Operating System
    flags := TFlags(hdr shr 24); // FLags
    if (FEXTRA in flags) then // extra field is present
    begin
      len := inStream.ReadWord; // extra field length
      inStream.Seek(len, soFromCurrent);// jump over extra field
      // parse subfields
      // |SI1|SI2|  LEN  |... LEN bytes of subfield data ...|
      // SI1 and SI2 provide a subfield ID
```

```
      // LEN gives the length of the subfield data, excluding the 4 initial
bytes
    end;
    if (FNAME in flags) then // the zero-terminated filename is present
    begin
      b := inStream.ReadByte;
      while b <> 0 do
      begin
        // sFilename := sFilename + char(b); // if filename is used
        b := inStream.ReadByte;
      end;
    end;
    if (FCOMMENT in flags) then // a zero-terminated comment is present
    begin
      b := inStream.ReadByte;
      while b <> 0 do
      begin
        // sComment := sComment + char(b); // if comment is used
        b := inStream.ReadByte;
      end;
    end;
    if (FHCRC in flags) then // there is a CRC16 for the header immediately
following the header
    begin
      crcH := crc16(0, pointer(inStream.Memory), inStream.Position); // get
crc16 checksum of the header
      crcHeader := inStream.ReadWord; // 2 bytes CRC16 for the header
      if crcH<>crcHeader then
        ;// header checksum mistake
    end;
    headerSize := inStream.Position;
    inStream.Seek(-8, soFromEnd);
    crcGZin := inStream.ReadDWord; // CRC32 (CRC-32)
    sizeGZin := inStream.ReadDWord; // ISIZE (Input SIZE)
    inStream.Size := inStream.Size-8; // cut the 4 byte crc32 and 4 byte
input size
  end
  else if (hdr and $00000078) = $00000078 then // zlib header
  begin
    streamType := zsZLib; // deflate format (with header)
    if (hdr and $00002000) = $00002000 then // FDICT preset dictionary
      headerSize := 6
    else
      headerSize := 2;
    inStream.Seek(-4, soFromEnd); // first byte is start of deflate header
    adler32in := SwapEndian(inStream.ReadDWord);
    inStream.Size := inStream.Size-4; // cut the 4 byte adler32 code
  end
  else
  begin
```

```
    streamType := zsRaw; // deflate format (is without header)
    headerSize := 0;
  end;

  // inflate raw stream
  inStream.Position := headerSize; // jump over header
  zstream.next_in := inStream.Memory + headerSize;
  zstream.avail_in := inStream.Size - headerSize;
  delta := (inStream.Size + 255) and not 255;
  if (streamType = zsGZip) then
    outStream.SetSize(sizeGZin)
  else
    outStream.SetSize(delta);
  zstream.next_out := outStream.Memory;
  zstream.avail_out := outStream.Size;
  if inflateInit2(zstream, RAW_WBITS) < 0 then Exit;
  while inflate(zstream, Z_NO_FLUSH) = Z_OK do
  begin
    outStream.SetSize(outStream.Size + delta);
    zstream.next_out := outStream.Memory + zstream.total_out;
    zstream.avail_out := delta;
  end;
  result := not (inflateEnd(zstream) < 0);
  outStream.SetSize(zstream.total_out);

  if result and (streamType = zsGZip) then // can check crc32 and size
  begin
    crc := crc32(0, Pointer(outStream.Memory), outStream.Size); // get
result crc32 checksum
    result := (crc = crcGZin) and (outStream.Size = sizeGZin); // compare
with input checksum and size
  end
  else if result and (streamType = zsZLib) then // can check adler32
checksum
  begin
    adler := adler32(0, Z_NULL, 0);
    adler := adler32(adler, Pointer(outStream.Memory), outStream.Size);
    result := (adler = adler32in);
  end;
  inStream.Position := 0; // goto start of source stream
  outStream.Position := 0; // goto start of result stream
end;

end.
```

Beispiel:

TestGzip.pas (/code/TestGzip.pas?mode=download) Pascal (2,13 kByte) 19.10.2016 15:50

```
program TestGzip;
```

```
{$mode objfpc}{$H+}

uses
  {$IFDEF UNIX}{$IFDEF UseCThreads}
  cthreads,
  {$ENDIF}{$ENDIF}
  Classes, SysUtils, GZIPUtils

  { you can add units after this };

procedure TestGZipUnGzip;
var
  MemoryStream, GZipStream: TMemoryStream;
  str1, str2: string;
  StartTime, EndTime: QWord;
  i: integer;
begin
  StartTime := GetTickCount64();

  // load gzip file to memory stream
  MemoryStream := TMemoryStream.Create;
  MemoryStream.LoadFromFile('browscap.ini');
  WriteLn('source size:' + IntToStr(MemoryStream.Size));

  // read memory stream to string
  MemoryStream.Seek(0, soFromBeginning);
  SetLength(str1, MemoryStream.Size);
  MemoryStream.ReadBuffer(PChar(str1)^, MemoryStream.Size);

  // gzip memory stream
  GZipStream := TMemoryStream.Create;
  StartTime := GetTickCount64();
  if zipStream(MemoryStream, GZipStream, zcDefault, zsGZip) then
  begin
    EndTime := GetTickCount64();
    WriteLn('gzip ok: ' + IntToStr(EndTime - StartTime) + ' ms');
  end
  else
    WriteLn('gzip failed');

  // save gzip memory stream to file
  GZipStream.SaveToFile('browscap.ini.gz');

  // ungzip memory stream
  MemoryStream.Clear;
  GZipStream.LoadFromFile('browscap.ini.gz');
  StartTime := GetTickCount64();
  if unzipStream(GZipStream, MemoryStream) then
  begin
    EndTime := GetTickCount64();
```

```
    WriteLn('ungzip ok: ' + IntToStr(EndTime - StartTime) + ' ms');
  end
  else
    WriteLn('ungzip failed');
  WriteLn('ungzip size:' + IntToStr(MemoryStream.Size));

  // save ungzip memory stream to file
  MemoryStream.SaveToFile('browscap2.ini');

  // read memory stream to string
  MemoryStream.Seek(0, soFromBeginning);
  SetLength(str2, MemoryStream.Size);
  MemoryStream.ReadBuffer(PChar(str2)^, MemoryStream.Size);

  // compare strings
  if str1 <> str2 then
    WriteLn('Strings do not match.')
  else
    WriteLn('Strings match.');

  FreeAndNil(MemoryStream);
  FreeAndNil(GZipStream);
end;
// size:22509806
// old:
//   gzip: 764 ms
//   ungzip: 281 ms
// new:
//   gzip: 593 ms
//   ungzip: 156 ms

begin
  TestGZipUnGzip;
  Writeln('Finished');
  Readln;
end.
```

Autor: Udo Schmal (#udo.schmal) , veröffentlicht: 18.06.2012, letzte Änderung: 06.09.2023