

## Chapter 15

# Basic Theory

## i-effect<sup>®</sup> \*ZIP Basics

### Algorithms

The “deflation Algorithm” used by zip and gzip is a variation of LZ77 (Lempel-Ziv 1977, see reference below). It finds duplicated strings in the input data. The second occurrence of a string is replaced by a pointer to the previous string, in the form of a pair (distance, length). Distances are limited to 32K bytes, and lengths are limited to 258 bytes. When a string does not occur anywhere in the previous 32K bytes, it is emitted as a sequence of literal bytes. (In this description, ‘string’ must be taken as an arbitrary sequence of bytes, and is not restricted to printable characters.)

Literals or match lengths are compressed with one Huffman tree, and match distances are compressed with another tree. The trees are stored in a compact form at the start of each block. The blocks can have any size (except that the compressed data for one block must fit in available memory). A block is terminated when zip determines that it would be useful to start another block with fresh trees.

Duplicated strings are found using a hash table. All input strings of length 3 are inserted in the hash table. A hash index is computed for the next 3 bytes. If the hash chain for this index is not empty, all strings in the chain are compared with the current input string, and the longest match is selected.

The hash chains are searched starting with the most recent strings, to favor small distances and thus take advantage of the Huffman encoding. The hash chains are singly linked. There are no deletions from the hash chains, the algorithm simply discards matches that are too old.

To avoid a worst-case situation, very long hash chains are arbitrarily truncated at a certain length, determined by a runtime option (1 to 9). So zip does not always find the longest possible match but generally finds a match which is long enough. Additionally, zip defers the selection of matches with a lazy evaluation mechanism.

After a match of length N has been found, zip searches for a longer match at the next input byte. If a longer match is found, the previous match is truncated to a length of one (thus producing a single literal byte) and the process of lazy evaluation begins again. Otherwise, the original match is kept, and the next match search is attempted only N steps later.

The lazy match evaluation is also subject to a runtime parameter. If the current match is long enough, zip reduces the search for a longer match, thus speeding up the whole process. If compression ratio is more important than speed, zip attempts a complete second search even if the first match is already long enough.

The lazy match evaluation is not performed for the fastest compression modes (level parameter 1 to 3). For these fast modes, new strings are inserted in the hash table only when no match was found, or when the match is not too long. This degrades the compression ratio but saves time since there are both fewer insertions and fewer searches.

according to *Jean-loup Gailly*  
*gzip@prep.ai.mit.edu*  
*jloup@gzip.org*

## Gzip File Format

The pkzip format imposes a lot of overheads in different headers, which are practical for an archive program, but not for compression of a single file. Gzip uses an easier structure: Figures are saved in the little endian file format and bit 0 is the least significant bit. A gzip file is a sequence of compressed members. Every member has the following structure:

2 Bytes	magic header 0x1f, 0x8b (037 \213)
1 Byte	compression method (0..7 reserved, 8 = deflate)
1 Byte	flags
	bit 0 set: file is probably ascii text file
	bit 2 set: extra field available
	bit 3 set: original file name available
	bit 4 set: file comment available
	bit 5 set: file is encoded
	bit 6,7: reserved
4 Bytes	date of file change in Unix file format
1 Byte	extra flags (depending on the method of compression)
1 Byte	System software on which the compression was conducted
2 Bytes	optional parts number (second part = 1)
2 Bytes	optional extra field length

? Bytes	optional extra field
? Bytes	optional original file name, X'00' in the end
? Bytes	optional file comment, X'00' in the end
12 Bytes	optional encryption header
? Bytes	compressed data
4 Bytes	crc32
4 Bytes	unsummarized file length modulo $2^{32}$

The file format was developed in order to

- conduct compression in "single-pass" without a reverse reading being necessary and
- without it being necessary or required to know the length of the uncompressed file or the amount of memory capacity on the output medium. If the input does not come from a regular disk file, the modification date will be set to the date of the beginning of compression.

Generally, the timestamp is practical if the gzip file is to be transmitted via a network. In this case, it is not sufficient to preserve only the file attributes. In the present case, the attributes of gzip are preserved during compression/decompression, a timestamp 0 will be ignored.

Bit 0 in the flags is an optional index that can be set through a short reading of the input file. When in doubt, the bit will be deleted in order to display binary data. On systems supporting different file formats for ASCII text and binary data, the decompressor can use this flag to choose the right format.

The extra field, if available, must consist of at least one subfield, each of them with the following format:

subfield ID	2 bytes
subfield size:	2 bytes (little endian format)

## Subfield Data

The subfield ID may consist of two characters of symbolic meaning. IDs with a second 0 byte are reserved for later use. Currently, the following IDs are defined:

Ap (0x41, 0x70): Apollo file type information

The subfield size is the size of subfield data and does not contain the ID or the size itself.

The field "extra field size" is the total length of the extra field, including the subfield ID and subfield size.

It needs to be possible to determine the end of the compressed data without depending on compression method or actual size of the compressed data. If the compressed data does not fit into a file (especially when using disks or tapes), every part will start with the head described above, but only the last part carries the information "crc32" and "uncompressed length". For such files compressed in various parts, a decompressor may send user requirements for additionally needed parts.

It is preferable, but not necessary, to decompress individual parts of a file independently from one another, in case a part is damaged. This will only be possible if the compression status is not taken over from one part to another.

If the file is compressed on a system using capital and lower case letters in its file system, the original file name must be changed into lower case letters. There will be no original file name if data is compressed before standard input.

Compression is always performed, even if the compressed file is slightly larger than the original. The worst case expansion is a few bytes for the gzip file header, plus 5 bytes every 32K block, or an expansion ratio of 0.015% for large files. Note that the actual number of used disk blocks almost never increases.

The encryption is a ZIP 1.9 encryption. The last byte of the decoded encryption header will be examined for the encryption checkup.

It must be 0. The timestamp of an encoded file can be set to 0 in order to avoid the derivation of the composition of the header.

according to *Jean-loup Gailly*  
*gzip@prep.ai.mit.edu*  
*jloup@gzip.org*

## Indication of References

[LZ77] Ziv J., Lempel A., „A Universal Algorithm for Sequential Data Compression“, IEEE Transactions on Information Theory, Vol. 23, No. 3, pp. 337-343.  
 APPNOTE.TXT documentation file in PKZIP 1.93a. It is available by  
 ftp in ftp.cso.uiuc.edu:/pc/exec-pc/pkz193a.exe [128.174.5.59]  
 Use „unzip pkz193a.exe APPNOTE.TXT“ to extract (note: unzip, not gunzip).

## ZIP File Format

In addition to the previously described gzip file format, i-effect® \*ZIP supports the ZIP archive file format, which can be found on many platforms. On Windows platforms for example, it became famous through products like WINZIP™.

A ZIP archive is a file in which compressed data of several files is stored. Appropriate software functions allow addition, deletion, extraction, or display of compressed contents.

i-effect® with its range of functions in the optional module \*ZIP is fully compatible with the common ZIP file format and, therefore, can exchange archived data with other platforms (e.g. WINZIP™).

Within an archive file (.ZIP file), different object types can be compressed. Currently, the following file types are supported: Physical files (PF-DTA), source files (SRC-PF), backup files (SAVF), files from IFS files systems and spooled files.

When restoring these object types with i-effect® \*ZIP on an IBM System i, the original object types will be received. Spooled files will be placed into output queues as actual spooled entries, compressed backup files become actual backup files and so on.

## EDIFACT Basics

The EDIFACT set of regulations provides a comprehensive basis of message definitions for many different commercial documents. The acronym EDIFACT stands for the claim to exchange routine business documents for administration, commerce, and transport in a standard format:

Electronic  
 Data  
 Interchange  
 For  
 Administration  
 Commerce and  
 Transport

The United Nations are a significant responsible for this standardization. Documents originating from the United Nations organizational structure are called UN/EDIFACT. In addition to these definitions, document formats from other institutions (e.g. Odette, representatives of the automotive industry), edited according to equal codification rules, exist. However, in their allover representation they do not correspond to the UN documents.

Because of its flexible database table logic, i-effect® is able to convert every document that is edited according to the EDIFACT syntax, inasmuch as an appropriate implementation code is provided. The documents have to be in accordance with the ISO 9735 syntax rules. A German version is available at the German Institute for Standardization (DIN) under publication number 16557.

Further information concerning EDIFACT in general and its practical application is provided by EDI consulting services and agencies or organizations involved in the standardization of EDIFACT documents, such as:

<b>DIN</b>	EDIFACT in general
<b>DEDIG</b>	EDIFACT related to practical application
<b>CCG / GS1</b>	Consumer goods management
<b>BSL</b>	Transport and Logistics
<b>VDA</b>	Automotive industry

and a lot more organizations and associations from various areas (e.g. textile, furniture, bank etc.)

## EDIFACT Rules for Conversion

In the following, formal and technical EDIFACT rules, which are relevant for conversion, are described.

The smallest information unit in EDIFACT is a data item or a data sub item, inasmuch as a composition of several data items is concerned. Hence, data items are components from which information is composed, either as a simple data item or as a composed data item, a so-called Composit. Every data item has certain features such as:

<b>Length</b>	e.g. 4
<b>Type of length</b>	fixed, variable
<b>Type of data</b>	numeric, alphanumeric, alphabetical
<b>Semantics</b>	content description

Concerning semantics, i.e. the description of the content of the field, EDIFACT diverges from usual encryption modes. The description on database fields or record type fields is based on the assumption that an unambiguous value is recorded in this field. One field is reserved for a telephone number, another field is reserved for a fax number. In EDIFACT, the unambiguous descriptions is realized by an interpretation of a further data field, a so-called Qualifier. In EDIFACT, the telephone and fax number example would be represented as follows:

<b>Contact number: 0226123990</b>	Qualifier: TE
<b>Contact number: 0226191845</b>	Qualifier: FX

The contact number is a data field for any type of communication connection to the described contact person; the used type of service (Telephone, fax, voice mail, telex, BTX screen text, Citytruf, GSM) is specified by the Qualifier. This type of representation was used to make sure that, in case of newly emerging means of communication, only the code list (list of all Qualifiers) needs to be updated, while the original form of the message is kept.

The composition of several data items and data item groups form a segment. Every segment contains a distinct, three-character description (segment tag). It describes a specific part of a document information, e.g.:

<b>NAD</b>	name and addresses
<b>DTM</b>	date/time/period
<b>MOA</b>	monetary amount
<b>TDT</b>	details of transport

Different segments can be combined into segment groups, inasmuch as they belong together with regard to contents. In some cases, a line item description in an order requires additional information, which is then combined into a segment group:

*segment group LIN-IMD-PIA-API...*

<b>LIN</b>	line item
<b>IMD</b>	item description
<b>PIA</b>	refund and reference
<b>API</b>	additional product ID

The composition of all segments and segment groups results in the actual message definition:

<b>INVOIC</b>	invoice
<b>ORDERS</b>	order
<b>IFTMIN</b>	instruction message

At the moment, more than 200 documents are defined in EDIFACT, which forms the basis for an intense and paperless exchange between companies, organizations, and other involved partners.

EDIFACT messages underlie a specified structure, whereas a distinction between service segment and generic segment is necessary.

EDIFACT assumes that the generated messages are transmitted from partner A to partner B via remote data processing. Therefore, special segments for the transmission of a physical file were defined. A physical file will be embraced by the segments

**UNB**  
...  
**UNZ**

and contains information for an addressee who is indicated in the UNB segment. Within this data interchange, different document types may occur, e.g. delivery notes and invoices. The document types are separated from each other with the help of the segments

**UNG**  
...  
**UNE**

The actual message stands between the segments

**UNH**  
...  
**UNT**

**The total structure of the data interchange may be as follows:**

<b>UNB</b>	beginning of data interchange
<b>UNG</b>	delivery notes
<b>UNH</b>	delivery note A
...	
<b>UNT</b>	end delivery note A

<b>UNH</b>	delivery note B
...	
<b>UNT</b>	end delivery note B
<b>UNE</b>	end delivery notes
<b>UNG</b>	invoices
<b>UNH</b>	invoice A
...	
<b>UNT</b>	end invoice A
<b>UNH</b>	invoice B
...	
<b>UNT</b>	end invoice B
<b>UNE</b>	end invoices
<b>UNZ</b>	end of data interchange

Every message the converter has to edit requires a clearly directed code conversion which constructs a relation (mapping) between the inhouse page and the EDIFACT page. This assignment has to be effected for every single data item within the segments and for every single type of segment in a segment group, inasmuch as the segment is assigned differently in different positions. The specific table options are described in the following chapter.

