ANSI®
X3.241-1994

American National Standard
for Information Systems –

Data Compression Method – Adaptive Coding with Sliding Window for Information Interchange

Secretariat

Computer and Business Equipment Manufacturers Association

Approved August 30, 1994

American National Standards Institute, Inc.

Contents

Page

Foreword ii

1 Scope and conformance 1

2 Normative references 1

3 Definitions 1

4 Algorithm identifier 1

5 Data format 2

Annexes

A Encoding overview 5

B Example compression encoding method 8

Foreword (This foreword is not part of American National Standard X3.241-1994.)

This standard specifies a lossless data compression method that is intended for general purposes. It contains features that make it particularly applicable to systems for recording information on interchangeable media.

This standard was developed by Technical Committee X3B5, by X3 project 882. The first draft was produced in November 1991. The second draft was produced in March 1992.

There are two annexes in this standard. Both are informative and are not considered part of this standard.

Requests for interpretation, suggestions for improvement or addenda, or defect reports are welcome. They should be sent to the X3 Secretariat, Computer and Business Equipment Manufacturers Association, 1250 Eye Street NW, Suite 200, Washington, DC 20005.

This standard was processed and approved for submittal to ANSI by the Accredited Standards Committee on Information Processing Systems, X3. Committee approval of this standard does not necessarily imply that all committee members voted for its approval. At the time it approved this standard, the X3 Committee had the following members:

American National Standard
for Information Systems –

Data Compression Method – Adaptive Coding with Sliding Window for Information Interchange

1 Scope and conformance

1.1 Scope

This standard specifies an encoding method for the lossless compression of binary data. This encoding method is known as Stacker LZS™.

1.2 Conformance

A data compression algorithm conforms to this standard if it satisfies all mandatory requirements.

2 Normative references

The following standard contains provisions which, through reference in this text, constitute a provision of this American National Standard. At the time of publication, the edition indicated was valid. All standards are subject to revision, and parties to agreements based on this American National Standard are encouraged to investigate the possibility of applying the most recent edition of the standard indicated below.

ISO/IEC 11576: 1994, *Information technology – Procedure for the registration of algorithms for the loss-less compression of data*

3 Definitions

The following definitions, listed in alphabetical order, apply to this standard.

3.1 compression ratio: The ratio of the number of bytes input into an encoding algorithm to the number of bytes produced by the encoding algorithm.

3.2 end marker: A unique bit pattern present in the output stream that represents the end of a block of compressed data.

3.3 history buffer: The compression encoding method requires reference to the most recently processed 2048 bytes of input data. These 2048 bytes are referred to as a *history buffer* or *sliding window*.

3.4 loss-less compression: A compression technique that allows for the complete reconstruction of the original input data stream without the introduction of any errors.

3.5 matching pattern: A multiple byte pattern residing in the history buffer that is identical to the source pattern.

3.6 raw byte token: A 9-bit pattern in the output stream that represents a single byte from the input stream.

3.7 sliding window: See *history buffer*.

3.8 source pattern: A multiple byte pattern entering the encoding algorithm that is identical to a corresponding matching pattern.

3.9 string token: A variable length bit pattern in the output stream that represents a source pattern.

4 Algorithm identifier

The numeric identifier of this encoding method in the International Register is 48.

5 Data format

5.1 Overview

The data compression encoding method is designed to support an adaptive string compression algorithm that can find redundant multiple byte patterns in the input data stream and replace them with shorter tokens in the compressed output data stream. The output data stream alone may be used to reconstruct the original input stream completely and exactly.

5.2 Principle of operation

The input data stream and the output data stream shall consist of a stream of bytes. Within a byte, the bits shall be arranged with bit b8 as the most significant, and bit b1 as the least significant.

The output data bytes are composed of a stream of fields with a variable number of bits in each field. The most significant bit of a field shall be placed into the most significant unused bit location of an output byte. The other bits of a field shall be placed in bit locations in an output byte by proceeding in sequence towards the least significant end of the output byte. The end marker (see 5.7) may be used to force the output data stream to a byte-boundary.

The output data stream consists of the following three field types:

– a raw byte token;

– a string token;

– an end marker.

The structure of these fields is described later. Each field may be present multiple times and may appear in any order. Each field shall be completed before a new field may begin.

The encoding algorithm shall maintain a 2048-byte history buffer. Each byte that enters the encoding algorithm from the input data stream shall be placed into the history buffer. When the history buffer becomes full, the newest bytes shall replace the oldest bytes. The history buffer shall always contain the last 2048 bytes that have entered the encoding algorithm since the last clearing of the history buffer.

The encoding algorithm shall monitor the input data stream and recognize multiple byte patterns that match identical multiple byte patterns within the history buffer. A multiple byte pattern entering the encoding algorithm is referred to as the source pattern. A multiple byte pattern already present in the history buffer, which matches the source pattern, is referred to as the matching pattern. The source pattern is replaced with a string token on the output bit stream. The string token represents the source pattern as described in 5.4.

There is no predefined maximum limit on the length of a source pattern. If the first byte of the matching pattern resides within the 2048-byte history buffer at the same time the first byte of the source pattern enters the history buffer, the multiple byte pattern can be represented correctly by the string token.

The smallest size of a source pattern is 2 bytes.

For each single byte in the input data stream that cannot be included in a source pattern, a raw byte token shall be output, (see 5.3.)

The end of a block of compressed data is uniquely identified by an end marker in the output stream, (see 5.7.)

5.3 Raw byte token

A raw byte token is a 9-bit pattern output by the encoding algorithm to represent a byte in the input stream that cannot be included within a source pattern. Bit b9 shall always be 0. Bits b1 through b8 shall be set to the same values as bits b1 through b8 of the input byte.

5.4 String token

A string token is a variable length bit pattern that represents a source pattern. A string token is composed of several bit-oriented fields. These fields are listed in the order that they are produced in the output stream:

— a single bit with a value of 1;

— an offset field (described in 5.5);

— a length field (described in 5.6).

5.5 Offset field

The offset field of a string token is a variable-length bit pattern that represents the distance in bytes within the history buffer from the first byte of the matching pattern to the first byte of the source pattern. The number of bytes is referred to as *offset.* The minimum value of *offset* is 1, which represents the byte that entered the history buffer just before the source pattern. The maximum value of *offset* is 2047.

If the value of *offset* is less than or equal to 127, an 8-bit pattern shall be generated. Bit b8 shall be a *1*. Bits b1 through b7 shall be the binary value of *offset.*

If the value of *offset* is greater than 127, a 12-bit pattern shall be generated. Bit b12 shall always be a *0*. Bits b11 through b1 shall be the binary value of *offset.*

5.6 Length field

The length field is a variable-length bit pattern that represents the length in bytes of the source pattern. The number of bytes is referred to as *length.* The minimum value of *length* is 2. The maximum value of *length* is unbounded.

If the value of *length* is less than or equal to 4, a 2-bit pattern shall be generated. These 2 bits shall be the binary value of:

(*length* – 2).

If the value of *length* is greater than 4, and less than or equal to 7, a 4-bit pattern shall be generated. Bits b3 and b4 shall always be a value of 1. Bits b1 and b2 shall be the binary value of:

(*length* – 5).

If the value of *length* is greater than 7, a variable-bit pattern shall be used. Multiple 4-bit patterns shall be generated with all bits set to *1*. The number of 4-bit patterns shall be:

(*N* + 1), where *N* is the integer result of ((*length* – 8) ÷ 15).

Then a 4-bit pattern shall be generated with a binary value of the remainder from the division operation.

5.7 End marker

The end marker is a unique 9-bit pattern that shall be generated at the end of a block of data. The history buffer may be optionally cleared at the end of a block of data. The minimum number of bytes that may be included in a block of data is *0*. The maximum number of bytes is unbounded.

Bits b8 and b9 shall be *1*s. Bits b1 through b7 shall be *0*s. Additional bits with a value of 0 shall be written to the output bit stream until an output byte-boundary is reached.

Annex A
(informative)

Example encoding algorithm

A.1 Encoding overview

This annex contains an example algorithm that may be used to encode an input stream of data. This algorithm consists of several processes. The process Encode is used to encode a complete block of data. This process will invoke other processes throughout its execution.

This algorithm assumes that two buffers are available that are infinite in size. Every new byte that enters the algorithm is inserted at the beginning of the history_buffer (while the entire contents of the history_buffer is moved by 1 byte to make space available). Every new byte is also inserted at the beginning of the holding_buffer, but the holding_buffer is emptied whenever a token is generated that represents the data stored in the holding_buffer. The most recent 2048 bytes in the history_buffer represent the history buffer as defined in 3.3. The holding_buffer represents the source pattern as defined in 3.7.

Other algorithms may be used to encode an input steam of data. Other algorithms need not require buffers of infinite size.

A.1.1 Encode

This process is used to compress an entire block of data. When this process is finished, the input byte stream will have been fully encoded and sent to the output byte stream.

```
Process = Encode.
While (input data exists).
    Read_byte.
    If ((there is no matching pattern in the history_buffer that exactly matches the source pat-
        tern in the holding_buffer, that also satisfies the condition that offset is less than 2047)).
        Output_token.
    Endif.
Endwhile.
Flush.
Endprocess.
```

Figure A.1 – Encode process

A.1.2 Read_byte

This process is invoked by the Encode process. It accepts a single byte from the input byte stream and pushes that byte onto both the history_buffer and the holding_buffer.

```
Process = Read_byte.
Get 8-bit byte from input stream.
Insert byte into history_buffer.
Insert byte into holding_buffer.
Endprocess.
```

Figure A.2 – Read_byte process

A.1.3 Output_token

This process is invoked by the Encode process. If a single byte is being processed, it will be output as a raw byte token. If multiple bytes are being processed, they will be output as a string token. A string token consists of an offset and a length.

```
Process = Output_token.
If (number of bytes in holding_buffer ≤ 2).
    Put single 0 bit to output bit stream.
    Put oldest byte in holding_buffer to output bit stream.
    Clear the oldest byte from the holding_buffer.
Elseif.
    Put single 1 bit to output bit stream.
    If (Offset ≤ 127).
        Put single 1 bit to output bit stream.
        Put 7-bit binary value of offset to output stream.
    Elseif.
        Put single 0 bit to output bit stream.
        Put 11-bit binary value of offset to output stream.
    Endif.
    Output_length.
    Clear all bytes from the holding_buffer except the newest byte.
Endif.
Endprocess.
```

Figure A.3 – Output_token process

A.1.4 Output_length

This process is invoked by the Output_token process. It will output the length portion of the string token being processed

```
Process = Output_length.
Set X to (number of bytes in holding_buffer – 1).
If ($X \le 4$).
  Put 2-bit binary value of ($X – 2$) to output stream.
Elseif.
  If ($X \le 7$).
    Put 2-bit pattern with all bits set to a 1 bit to output stream.
    Put (2-bit binary value of ($X – 5$) to output stream.
  Elseif.
    Put 4-bit pattern with all bits set to a 1 bit to output stream.
    Set X to ($X – 8$).
    While ($X \ge 15$).
      Put 4-bit pattern with all bits set to a 1 bit to output stream.
      Set X to ($X – 15$).
    Endwhile.
    Put 4-bit binary value of X to output stream.
  Endif.
Endif.
Endprocess.
```

Figure A.4 – Output_length process

A.1.5 Offset

This process is invoked by the Output_token process. This process will calculate and return the offset portion of the string token to the Output_token process.

```
Process = Offset.
Return the value of (the distance in bytes within the history_buffer from the first
    byte of the source pattern to the first byte of the matching pattern).
Endprocess.
```

Figure A.5 – Offset process

A.1.6 Flush

This process is invoked by the Encode process. This process will force any pending token to be output, followed by the end marker.

```
Process = Flush.
While (number of bytes in holding_buffer > 0).
    Output_token.
Endwhile.
Put 9-bit pattern with b8 and b9 set to 1s and bits b1 through b7 set to 0s to output stream.
If (desired to clear the history).
    Clear all bytes from the history_buffer.
Endif.
Endprocess.
```

Figure A.6 – Flush process

Annex B
(informative)

Example compression encoding

Table B.1 demonstrates an example encoding output based on a given input byte stream. Within table B.1, time runs down the page.

Table B.1 – Example encoding output

Input byte stream	Output bit stream	Comments
A	–	Source pattern requires at least 2 bytes to match
B	001000001_2	No matching pattern for AB, output A as raw byte token
A	001000010_2	No matching pattern for BA, output B as raw byte token
A	001000001_2	No matching pattern for AA, output A as raw byte token
A	–	Matching pattern found for AA, wait for possible longer pattern
A	–	Matching pattern found for AAA, wait for possible longer pattern
A	–	Matching pattern found for AAAA, wait for possible longer pattern
A	–	Matching pattern found for AAAAA, wait for possible longer pattern
C	1100000011100_2	No matching pattern for AAAAAC, output AAAAA as string token with an offset of 1 and length of 5
A	001000011_2	No matching pattern for CA, output C as raw byte token
B	–	Matching pattern found for AB, wait for possible longer pattern
A	–	Matching pattern found for ABA, wait for possible longer pattern
B	11000100101_2	No matching pattern for ABAB, output ABA as string token with an offset of 9 and length of 3
A	–	Matching pattern found for BA, wait for possible longer pattern
B	–	Matching pattern found for BAB, wait for possible longer pattern
A	–	Matching pattern found for BABA, wait for possible longer pattern
end	11000001010_2 110000000_2	End of block, output BABA as string token with an offset of 2 and length of 4, then output end marker

Summary for this example:

Input byte stream (16 bytes, specified in base 16): 41 42 41 41 41 41 41 41 43 41 42 41 42 41 42 41

Output byte stream (10 bytes, specified in base 16): 20 90 88 38 1C 21 E2 5C 15 80