Compression Algorithm Doc V1

For suggestions/feedback/corrections: Contact Dan McCarthy

MIO0 Introduction:

The MIO0 compression algorithm is based on the LZ family of compression. It makes use of a dictionary/sliding window design. MIO0 splits data into three categories as it processes the file data.

The first is non-compressed data, the actual letters/bytes of data found in the file. As the file is processed the algorithm cuts down on repeated characters or phrases it files. At least one of every character/byte found in the file will be in the non-compressed category, but ideally repeats will be removed. When repeated characters or phrases are found, the algorithm documents the position and how many characters were found to repeat. It will then put the length/offset data into the compressed-data section. Finally, for when the file is being decompressed, we keep track of which order the compressed data and non-compressed data are written out with a section of bits. A bit set to 1 represents non-compressed data, and a bit set to 0 represents compressed data.

The order of the data in the file from beginning to end goes as such:

Header Data | Layout Bits | Compressed Data | Non-Compressed Data

The format of the header data (16 bytes total):

4 byte "Magic Number" or "Version Number": (String:) "MIO0" (Hexadecimal:) 4D 49 4F 30
4 byte Decompressed Size of File
4 byte Offset to Compressed Data in File
4 byte Offset to Non-Compressed Data in File

The format of the compressed data (2 bytes each):

Each segment of the compressed data consists of 2 bytes, representing the offset from which data will be copied on decompression, and the length of how many bytes need to be copied (More detail in Decompression section). The length value is 4 bits in length, and the offset value is 12 bits in length. (With the length being derived from the first 4 bits of the first byte, and the offset being the last 4 bits of the second byte and the 8 bits of the second byte.)

For a sample from the Example File, we have the bytes 20 (0100 0000) 11 (0001 0001):

The first 4 bits of 20 (0100) make up the length. Therefore, the length is 2 in decimal.
The last 4 bits of 20 (0000) and the full 8 bits of 11 (0001 0001) make up the offset.
Therefore the offset is 11 in decimal.

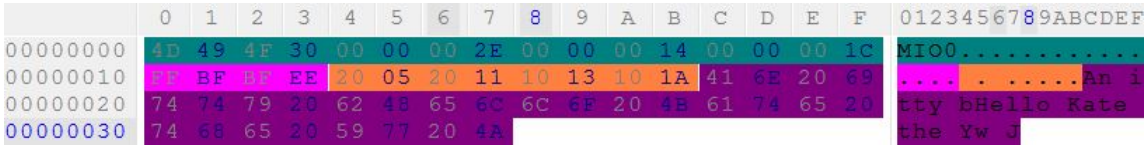
Length: 2 (0010)
Offset: 11 (0000 0001 0001)

Note: On compression of the data, the value 3 is subtracted from the length value, and the value 1 is subtracted from the offset. This must be added back on during decompression.

The format of the non-compressed data (1 byte each):

The non-compressed data is presented in order of use, with the first byte being the first character/byte to be copied into the new file on decompression, and the last being the final. They begin at the Non-compressed data offset from the header data and continue until the end of the file.

Example File (Big Endian):



	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	0123456789ABCDEF
00000000	4D	49	4F	30	00	00	00	2E	00	00	00	14	00	00	00	1C	MIO0.....
00000010	FF	BF	BF	EE	20	05	20	11	10	13	10	1A	41	6E	20	69	....An i
00000020	74	74	79	20	62	48	65	6C	6C	6F	20	4B	61	74	65	20	tty bHello Kate
00000030	74	68	65	20	59	77	20	4A									the Yw J

Example File Breakdown:

The initial 16 bytes are (cyan) the header data:

- **The first 4 bytes (4D 49 4F 39) spell out "MIO0," the version number for the format.**
- **The next 4 bytes (00 00 00 2E) represent the size of the string/data before compression ("An itty bitty Hello Kitty ate the Yellow Jello")**
- **The following 4 bytes (00 00 00 14) represent the beginning of the Compressed Data (orange) 0x14.**
- **The last 4 bytes of the header data (00 00 00 1C) represent the beginning of Non-Compressed Data (Purple) 0x1C.**

The following 4 bytes (Magenta) are the control/layout bits:

(FF) (BF) (BF) (EE)
(1111 1111) (1011 1111) (1011 1111) (1110 1110)

Bits read one by one from left to right. On decompression a new file is created and each layout bit is looped through. If the indexed bit is 1, the first character/byte from non-compressed data is copied into the new file. Eventually when a 0 is hit, the first 2 bytes of compressed data are read and an offset and length are retrieved from them. The length amount of Characters/bytes from that offset of the new file are then copied to the end of the new file. This is repeated until the new file is the same size as the decompressed size from the header.

The next 8 bytes (Orange) are the Compressed Data:

First 2:
Hex/Bin: 20 05 (0010 0000 0000 0101):
Length: 2 (0010) Offset: 5 (0000 0000 0101)

Second 2:
Hex/Bin: 20 11 (0010 0000 0001 0001):
Length 2 (0010) Offset: 17 (0000 0001 0001)

Third 2:
Hex/Bin: 10 13 (0001 0000 0001 0011)
Length: 1 (0001) Offset: 19 (0000 0001 0011)

Last 2:
Hex/Bin: 10 1A (0001 0000 0001 1010)
Length: 1 (0001) Offset: 26 (0000 0001 1010)

The last 28 bytes of the file (Purple) are the Non-Compressed Data:

Hex: 41 6E 20 69 74 74 79 20 62 48 65 6C 6C 6F 20 4B 61 74 65 20 74 68 65 20 59 77 20 4A
String: "An itty bHello Kate the Yw J"

Decompressed String:
"An itty bitty Hello Kitty ate the Yellow Jello"

YAY0 Introduction:

The YAY0 compression algorithm is very similar to that of MIO0. It is also based upon the LZ family of compression algorithms and is nearly identical to the implementation of MIO0 with slight modification. Due to how MIO0 stores it's compressed data, there are very sharp limits to how long a set of characters being compressed can be (the stored length can only be 4 bits in size). YAY0 helps get around this issue by having the ability to store the length as 4 bits if the size is small enough to fit within 4 bits, or will otherwise store an additional entire byte that represents the length value in the non-compressed data.

Note: 1 is still subtracted from the offset. 2 is subtracted from length if length is less than 18, otherwise 18 is subtracted from the additional byte read, and that becomes the length.

The order of the data in the file from beginning to end goes as such:

Header Data | Layout Bits | Compressed Data | Non-Compressed Data

The format of the header data (16 bytes total):

4 byte "Magic Number" or "Version Number": (String:) "Yay0" (Hexadecimal:) 0x59 0x61 0x79 0x30
4 byte Decompressed Size of File
4 byte Offset to Compressed Data in File
4 byte Offset to Non-Compressed Data in File

The format of the compressed data (2 bytes each):

Each segment of the compressed data consists of 2 bytes, representing the offset from which data will be copied on decompression, and the length of how many bytes need to be copied (More detail in Decompression section). The length value is 4 bits in length, and the offset value is 12 bits in length. (With the length being derived from the first 4 bits of the first byte, and the offset being the last 4 bits of the second byte and the 8 bits of the second byte.) Because the length value can only be 4 bits, the length value can normally only go as high as 17 (15 from 4 bits, but 2 is added to this value for additional range). If the length is higher than 17, the 4 length bits are instead set to 0, and the length value is made into its own byte. This byte value has 18 added to it and is placed in the non-compressed data section.

For a sample from the Example File, we have the bytes 0x20 (0100 0000) 0x11 (0001 0001):

The first 4 bits of 0x20 (0100) make up the length. Therefore, the length is 2 in decimal. The last 4 bits of 0x20 (0000) and the full 8 bits of 0x11 (0001 0001) make up the offset. Therefore the offset is 11 in decimal.

Length: 2 (0010)
Offset: 17 (0000 0001 0001)

In this case, the length value is less than 18 so it uses the 4 bits.

For another sample from the Example File, we have bytes 0x00 (0000) 0x2E (0010 1110):

In the example, the entire sentence is repeated. Therefore, the last of the compressed data will have a length much greater than 18 and will require use of this extra length byte.

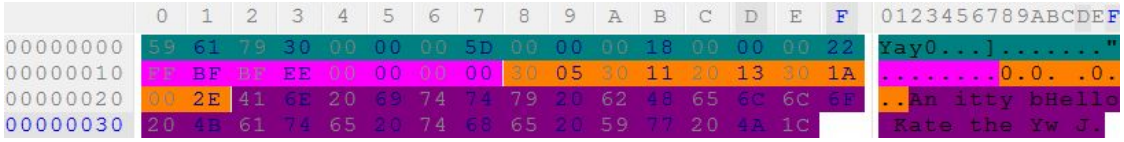
Length: 0 (0000)
Offset: 46 (0010 1110)

Because the length is 0, we can not use this. This lets us know we must instead read the length from a byte from the non-compressed data. This would be the next non-compressed byte that has not been read. In this case, since this is the last compressed data, the length byte is the last byte of the file 0x1C (28). Because we read it from a byte, we add 18 to it, giving us 0x2E (46).
Note: On compression of the data, the value 2 is subtracted from the length value, and the value 1 is subtracted from the offset. This must be added back on during decompression.

The format of the non-compressed data (1 byte each):

The non-compressed data is presented in order of use, with the first byte being the first character/byte to be copied into the new file on decompression, and the last being the final. They begin at the Non-compressed data offset from the header data and continue until the end of the file. Additionally, when the compressed data has a length higher than 18, the length can be encoded into a byte that is then placed in the non-compressed data.

Example File (Big Endian):



Example File Breakdown:

The initial 16 bytes are (cyan) the header data:

- The first 4 bytes (59 61 79 30) spell out "Yay0," the version number for the format.
- The next 4 bytes (00 00 00 5D) represent the size of the string/data before compression ("An itty bitty Hello Kitty ate the Yellow Jello An itty bitty Hello Kitty ate the Yellow Jello")
- The following 4 bytes (00 00 00 18) represent the beginning of the Compressed Data (orange) 0x18.
- The last 4 bytes of the header data (00 00 00 22) represent the beginning of Non-Compressed Data (Purple) 0x22.

The following 4 bytes (Magenta) are the control/layout bits:

(FF)	(BF)	(BF)	(EE)	(00)	(00)	(00)	(00)
(1111 1111)	(1011 1111)	(1011 1111)	(1110 1110)	(0000 0000)	(0000 0000)	(0000 0000)	(0000 0000)

Bits read one by one from left to right. On decompression a new file is created and each layout bit is looped through. If the indexed bit is 1, the first character/byte from non-compressed data is copied into the new file. Eventually when a 0 is hit, the first 2 bytes of compressed data are read and an offset and length are retrieved from them. The length amount of Characters/bytes from that offset of the new file are then copied to the end of the new file. This is repeated until the new file is the same size as the decompressed size from the header.

The next 8 bytes (Orange) are the Compressed Data:

First 2:
Hex/Bin: 30 05 (0011 0000 0000 0101):
Length: 3 (0011) Offset: 5 (0000 0000 0101)

Second 2:
Hex/Bin: 30 11 (0011 0000 0001 0001):
Length 3 (0011) Offset: 17 (0000 0001 0001)

Third 2:
Hex/Bin: 20 13 (0010 0000 0001 0011)
Length: 2 (0010) Offset: 19 (0000 0001 0011)

Fourth 2:
Hex/Bin: 30 1A (0011 0000 0001 1010)
Length: 3 (0011) Offset: 26 (0000 0001 1010)

Last 2: ****Unique, length 0, meaning it requires reading of a byte from non-compressed data for length***
Hex/Bin: 00 2E (0000 0000 0001 1010)
Length: 0 (0000) Offset: 46 (0000 0010 1110)

The last 29 bytes of the file (Purple) are the Non-Compressed Data:

Hex: 41 6E 20 69 74 74 79 20 62 48 65 6C 6C 6F 20 4B 61 74 65 20 74 68 65 20 59 77 20 4A 1C

String:
"An itty bHello Kate the Yw J."

Decompressed String:
"An itty bitty Hello Kitty ate the Yellow Jello An itty bitty Hello Kitty ate the Yellow Jello"

(Because YAY0's advantage over MIO0 is it can compress greater than 18 bytes at once, we can demonstrate this by repeating the entire previous example, which will only lead to 2 extra compressed data bytes and 2 extra non-compressed data bytes.

YAZ0 Introduction:

The YAZ0 compression is an alteration of the YAY0 compression algorithm as YAY0 is to MIO0. YAZ0 keeps the variable sizes the same as YAY0, but instead changes how the data is ordered in the compressed file. Where MIO0 and YAY0 need to jump between the offsets of the non-compressed data, the compressed data, and the layout/control bits, as well as needing to increment those offsets as it goes, YAZ0 does not. While YAZ0 requires the same three categories of data, the layout/control bits, the compressed data, and the non-compressed data, it does not place them in their own three areas within the file. Instead, 8 layout/control bits (one full byte) are written, and is followed by the non-compressed data and compressed data in the order that they are needed (changes depending on the specific layout/control bits.) After the 8 operations worth of data is read, it is followed by the next 8 bits until the entire file is accounted for.

Note: Same as YAY0, 2 is subtracted from length if length is less than or equal to 18, otherwise 18 is subtracted from the additional byte read, and that becomes the length. Except the additional byte is written in the order it is required, and is not read from an offset location.

<u>The order of the data in the file:</u>

Header Data | First Layout Bits | Can Change | Can Change | Second Layout Bits | … repeat

Depending on the order of the bits of the layout/control bits changes the order in which compressed data and non compressed data are ordered. The data required for 8 operations (one for each bit) follow the 8 layout bits, then another 8 layout bits follow that, as well as the data required until the file is accounted for.

Layout bits (0000 0000): 8 pairs of compressed data
Layout bits (1111 1111): 8 bytes of non-compressed data
Layout bits (1010 1010): Alternating non-compressed data and compressed data.

<u>The format of the header data (16 bytes total):</u>

4 byte "Magic Number" or "Version Number": (String:) "Yaz0" (Hexadecimal:) 0x59 0x61 0x7A 0x30
4 byte Decompressed Size of File
4 byte Offset to Compressed Data in File
4 byte Offset to Non-Compressed Data in File

<u>The format of the compressed data (2 bytes each):</u>

Each segment of the compressed data consists of 2 bytes, representing the offset from which data will be copied on decompression, and the length of how many bytes need to be copied (More detail in Decompression section). The length value is 4 bits in length, and the offset value is 12 bits in length. (With the length being derived from the first 4 bits of the first byte, and the offset being the last 4 bits of the second byte and the 8 bits of the second byte.) Because the length value can only be 4 bits, the length value can normally only go as high as 17 (15 from 4 bits, but 2 is added to this value for additional range). If the length is higher than 17, the 4 length bits are instead set to 0, and the length value is made into its own byte. This byte value has 18 added to it and is placed in the non-compressed data section.

<u>For a sample from the Example File, we have the bytes 0x20 (0100 0000) 0x11 (0001 0001):</u>

The first 4 bits of 0x20 (0100) make up the length. Therefore, the length is 2 in decimal. The last 4 bits of 0x20 (0000) and the full 8 bits of 0x11 (0001 0001) make up the offset. Therefore the offset is 11 in decimal.

Length: 2 (0010)
Offset: 17 (0000 0001 0001)

In this case, the length value is less than 18 so it uses the 4 bits.

<u>For another sample from the Example File, we have bytes 0x00 (0000) 0x2E (0010 1110):</u>

In the example, the entire sentence is repeated. Therefore, the last of the compressed data will have a length much greater than 18 and will require use of this extra length byte.

Length: 0 (0000)
Offset: 46 (0010 1110)

Because the length is 0, we can not use this. This lets us know we must instead read the length from a byte from the non-compressed data. This would be the next non-compressed byte that has not been read. In this case, since this is the last compressed data, the length byte is the last byte of the file 0x1C (28). Because we read it from a byte, we add 18 to it, giving us 0x2E (46).
Note: On compression of the data, the value 2 is subtracted from the length value, and the value 1 is subtracted from the offset. This must be added back on during decompression.

The format of the non-compressed data (1 byte each):

The non-compressed data is presented in order of use, with the first byte being the first character/byte to be copied into the new file on decompression, and the last being the final. They begin at the Non-compressed data offset from the header data and continue until the end of the file. Additionally, when the compressed data has a length higher than 18, the length can be encoded into a byte that is then placed in the non-compressed data.

Example File (Big Endian):

Example File Breakdown:

The initial 16 bytes are (cyan) the header data:

- The first 4 bytes (59 61 7A 30) spell out "Yaz0," the version number for the format.
- The next 4 bytes (00 00 00 5D) represent the size of the string/data before compression ("An itty bitty Hello Kitty ate the Yellow Jello An itty bitty Hello Kitty ate the Yellow Jello")
- The following 8 bytes are no longer required as there are no offsets. The file is now read in order. Because of this, these 8 bytes are now set to 0.

The following byte (Magenta) are the first set of control/layout bits:

(FF) (1111 1111) [non-compressed data][non-compressed data][non-compressed data][non-compressed data]
[non-compressed data][non-compressed data][non-compressed data][non-compressed data]

Bits read one by one from left to right. On decompression a new file is created and each layout bit is looped through. If the indexed bit is 1, the first character/byte from non-compressed data is copied into the new file, which will be the next byte of the file. Eventually when a 0 is hit, the next 2 bytes of compressed data are read and an offset and length are retrieved from them. The length amount of Characters/bytes from that offset of the new file are then copied to the end of the new file. This is repeated until the new file is the same size as the decompressed size from the header.

The next 8 bytes of the file (Purple) are Non-Compressed Data:

Hex: 41 6E 20 69 74 74 79 20
String:
"An itty "

The first 8 layout/control bits were all 1, therefore the bits are followed by 8 non-compressed bytes

The next byte (Magenta) are the next set of Control/Layout bits:

(BF)(1011 1111) [non-compressed data][compressed data][non-compressed data][non-compressed data]
[non-compressed data][non-compressed data][non-compressed data][non-compressed data]

All but one of the bits in this case are 1, therefore 7 bytes will be non-compressed data, and 1 pair of compressed data bytes for the singular 0 bit. The order of the bits decide the order of these bytes. In this case, the first byte will be non-compressed, followed by the compressed pair, and the rest of the non-compressed data follows after that.

The next byte (Purple) is a non-compressed byte:

Hex: 62
String:
"B"

The next 2 bytes (Orange) are first the set of Compressed Data:

Hex/Bin 30 05 (0011 0000 0000 00101)
Length: 3 (0011) Offset: 5 (0000 0000 0101)

The next 7 bytes (Purple) are non-compressed Data:

Hex: 48 65 6C 6C 6F 20
String:
"Hello "

…

This pattern repeats, 8 layout bits, followed by the data needed for those 8 operations in the order they are required, then the next 8 layout bits followed by the data required for those 8 operations. This is repeated until the full file is accounted for.

…

(Note: Because the majority of the file is essentially the same as above, with different orderings, we will skip to the last set as it is the only one that is unique)

The next byte (Magenta) are the next set of Control/Layout bits:

(EE)(1110 1110) [non-compressed data][non-compressed data][non-compressed data][compressed data]
[non-compressed data][non-compressed data][non-compressed data][compressed data]

The next three bytes (Purple) are non-compressed bytes:

Hex: 65 20 59
String:
"e Y"

The next 2 bytes (Orange) are first the set of Compressed Data:

Hex/Bin 30 05 (0011 0000 0000 00101)
Length: 3 (0011) Offset: 5 (0000 0000 0101)

The next three bytes (Purple) are non-compressed bytes:

Hex: 77 20 4A
String:
"w J"

The next 2 bytes (Orange) are first the set of Compressed Data:

Hex/Bin 30 1A (0011 0000 0001 1010)
Length: 3 (0011) Offset: 26 (0000 0001 1010)

The next byte (Purple) is a non-compressed byte:

Hex: 00
String:
"."

(Here are where things get interesting.)

The next 2 bytes (Orange) are first the set of Compressed Data:

Hex/Bin 00 2E (0000 0000 0010 1110)
Length: 0 (0000) Offset: 46 (0000 0010 1110)

(For the first time in the file, the length is "0." This represents the length being a value of 18 or higher, which does not fit in the 4 bit length limit. Therefore, the length had to be written as its own byte. Because it is the byte we need next, it will be the byte following the compressed data we have just read in. (Also will be the last byte of the file)

The final byte (Purple)is a non-compressed byte:

Hex: 1C
String:
"."
This byte is not a part of the string, but a length value byte. We read this byte, subtract 18 from it, and it becomes our new length value to read the last compressed data. In this case, the data being copied is the entire string over again. We were able to compress the entire repetition in 2 additional bytes and a bit.

Other resources:

MIO0 Documentation:
http://wiki.origami64.net/super_mario_64/mio0
http://origami64.net/showthread.php?tid=113 **(Very similar, but has very useful example in .txt file)**

MIO0 Source Code:
https://github.com/queueRAM/sm64tools/blob/master/libmio0.c
http://shygoo.net/mio0/src/mio0.c

YAY0 Documentation/Source Code:
http://hitmen.c02.at/files/yagcd/yagcd/chap16.html
https://github.com/pho/WindViewer/wiki/Yaz0-and-Yay0

YAZ0 Documentation:
http://www.amnoid.de/gc/yaz0.txt
http://wiki.tockdom.com/wiki/Yaz0
http://hitmen.c02.at/files/yagcd/yagcd/chap16.html

YAZ0 Source Code:
https://github.com/Syroot/NintenTools.Yaz0
https://github.com/RenolY2/yaz0-decode-encode
https://github.com/pho/WindViewer/wiki/Yaz0-and-Yay0
