

# Package: bitstreamio (via r-universe)

January 15, 2025

**Type** Package

**Title** Read and Write Bits from Files, Connections and Raw Vectors

**Version** 0.1.0

**Maintainer** Mike Cheng <mikefc@coolbutuseless.com>

**URL** <https://github.com/coolbutuseless/bitstreamio>

**BugReports** <https://github.com/coolbutuseless/bitstreamio/issues>

**Description** Bit-level reading and writing are necessary when dealing with many file formats e.g. compressed data and binary files. Currently, R connections are manipulated at the byte level. This package wraps existing connections and raw vectors so that it is possible to read bits, bit sequences, unaligned bytes and low-bit representations of integers.

**License** MIT + file LICENSE

**Encoding** UTF-8

**RoxygenNote** 7.3.2

**Suggests** knitr, rmarkdown, testthat (>= 3.0.0)

**Config/testthat/edition** 3

**VignetteBuilder** knitr

**Repository** <https://coolbutuseless.r-universe.dev>

**RemoteUrl** <https://github.com/coolbutuseless/bitstreamio>

**RemoteRef** HEAD

**RemoteSha** 036447f7ca01cb05668cea92a542fe6fcee5a4c4

## Contents

assert_bs . . . . .	2
bits_to_raw . . . . .	3
bits_to_uint . . . . .	3
bs_advance . . . . .	4
bs_align . . . . .	5

bs_flush . . . . .	5
bs_is_aligned . . . . .	6
bs_open . . . . .	7
bs_peek . . . . .	8
bs_read_bit . . . . .	8
bs_write_bit . . . . .	9
bs_write_byte . . . . .	10
bs_write_sint_exp_golomb . . . . .	11
bs_write_uint . . . . .	11
bs_write_uint_exp_golomb . . . . .	12
is_bs . . . . .	13
pad_bits . . . . .	14
sint_to_exp_golomb_bits . . . . .	14
uint_to_exp_golomb_bits . . . . .	15

**Index** **16**

---

assert_bs	<i>Test if an object is a bitstream object and fail if it is not</i>
-----------	--

---

**Description**

Test if an object is a bitstream object and fail if it is not

**Usage**

```
assert_bs(x)
```

**Arguments**

x	object to test
---	----------------

**Value**

None

**Examples**

```
raw_vec <- as.raw(1:3)
bs <- bs_open(raw_vec, 'r')
assert_bs(bs)
bs_close(bs)
```

---

bits\_to\_raw                      *Convert between logical vector of bits and raw vector*

---

**Description**

Convert between logical vector of bits and raw vector

**Usage**

```
bits_to_raw(bits, msb_first = TRUE)
```

```
raw_to_bits(x, msb_first = TRUE)
```

**Arguments**

bits	Logical vector of bit values. Length must be a multiple of 8
msb_first	MSB first? Default: TRUE
x	Byte values. Integer vectors will be truncated to 8 bits before output. Numeric vectors will be rounded to integers and then truncated to 8 bits. Raw vectors preferred.

**Value**

Logical vector of bit values or a raw vector.

**Examples**

```
bits <- raw_to_bits(c(0, 4, 21))
bits
bits_to_raw(bits) |> as.integer()
```

---

bits\_to\_uint                      *Convert between bits and unsigned integers*

---

**Description**

Convert between bits and unsigned integers

**Usage**

```
bits_to_uint(bits, nbits = NULL)
```

```
uint_to_bits(x, nbits)
```

**Arguments**

bits	logical vector of bit values in MSB first order
nbits	number of bits per integer. If NULL, then bits is assumed to represent a single integer value. If not NULL, then the number of values in bits must be a multiple of nbits
x	vector of unsigned integers

**Value**

logical vector of bit values of vector of unsigned integers

**Examples**

```
bits <- uint_to_bits(c(1, 2, 3), nbits = 3)
bits
bits_to_uint(bits, nbits = 3)
```

---

bs_advance	<i>Advance bitstream</i>
------------	--------------------------

---

**Description**

Advance bitstream

**Usage**

```
bs_advance(bs, n)
```

**Arguments**

bs	Bistream connection object created with bs_open()
n	number of bits to advance

**Value**

Bitstream connection returned invisibly

**Examples**

```
raw_vec <- as.raw(1:3)
bs <- bs_open(raw_vec, 'r')
bs_is_aligned(bs)
bs_advance(bs, 4)
bs_is_aligned(bs)
bs_read_bit(bs, 8)
bs_is_aligned(bs)
bs_close(bs)
```

---

bs_align	<i>Align the bitstream to the given number of bits - relative to start of bitstream</i>
----------	---

---

**Description**

Align the bitstream to the given number of bits - relative to start of bitstream

**Usage**

```
bs_align(bs, nbits = 8L, value = FALSE)
```

**Arguments**

bs	Bistream connection object created with bs_open()
nbits	number of bits of alignment w.r.t start of bitstream. Default: 8
value	bit fill value. Either TRUE or FALSE. Default FALSE

**Value**

Bitstream connection returned invisibly

**Examples**

```
bs <- bs_open(raw(), 'w')
bs_write_bit(bs, c(TRUE, FALSE, TRUE))
bs_is_aligned(bs, 8)
bs_align(bs, nbits = 8)
bs_is_aligned(bs, 8)
output <- bs_close(bs)
output
```

---

bs_flush	<i>Flush bits in the buffer</i>
----------	---------------------------------

---

**Description**

This is called internally to flush bitstream buffers to the underlying R connection.

**Usage**

```
bs_flush(bs)
```

**Arguments**

bs	Bistream connection object created with bs_open()
----	---

**Value**

Bitstream connection returned invisibly

**Examples**

```
bs <- bs_open(raw(), 'w')
bs_write_bit(bs, c(TRUE, FALSE, TRUE))
bs_align(bs, nbits = 8)
bs_flush(bs)
output <- bs_close(bs)
output
```

---

bs_is_aligned	<i>Is the current bit connection aligned at the given number of bits for reading/writing?</i>
---------------	---

---

**Description**

Is the current bit connection aligned at the given number of bits for reading/writing?

**Usage**

```
bs_is_aligned(bs, nbits = 8)
```

**Arguments**

bs	Bistream connection object created with bs_open()
nbits	number of bits of alignment w.r.t start of bitstream. Default: 8

**Value**

logical. TRUE if stream location is currently aligned to the specified number of bits, otherwise FALSE

**Examples**

```
bs <- bs_open(raw(), 'w')
bs_write_bit(bs, c(TRUE, FALSE, TRUE))
bs_is_aligned(bs, 8)
bs_align(bs, nbits = 8)
bs_is_aligned(bs, 8)
output <- bs_close(bs)
output
```

---

bs_open	<i>Open/close a bitstream</i>
---------	-------------------------------

---

## Description

Open/close a bitstream

## Usage

```
bs_open(con, mode, msb_first = TRUE, flush_threshold = 1024 * 8)
```

```
bs_close(bs, verbosity = 0)
```

## Arguments

con	A vector of raw values or an R connection (e.g. <code>file()</code> , <code>url()</code> , etc)
mode	Bitstream mode set to read or write? One of 'r', 'w', 'rb', 'wb'.
msb_first	Should the output mode be Most Significant Bit first? Default: TRUE
flush_threshold	Threshold number of bits at which the buffered data will be automatically written to the connection. Default: 8192 bits (1024 bytes). Note: Use <code>bs_flush()</code> to write out the buffer at any time. All bits are automatically written out when <code>bs_close()</code> is called.
bs	Bistream connection object created with <code>bs_open()</code>
verbosity	Verbosity level. Default: 0

## Value

`bs_open()` returns a `bitstream` connection object. When the connection is a raw vector and mode = 'w', `bs_close()` returns the final state of the raw vector; in all other cases `bs_close()` does not return a value.

## Examples

```
raw_vec <- as.raw(1:3)
bs <- bs_open(raw_vec, 'r')
assert_bs(bs)
bs_close(bs)
```

---

bs_peek	<i>Peek at bits from a bitstream i.e. examine bits without advancing bitstream</i>
---------	--

---

**Description**

Peek at bits from a bitstream i.e. examine bits without advancing bitstream

**Usage**

```
bs_peek(bs, n)
```

**Arguments**

bs	Bistream connection object created with bs_open()
n	number of bits to peek.

**Value**

logical vector of bit values

**Examples**

```
raw_vec <- as.raw(1:3)
bs <- bs_open(raw_vec, 'r')
bs_peek(bs, 4)
stopifnot(bs_is_aligned(bs))
bs_close(bs)
```

---

bs_read_bit	<i>Read bits from a bitstream</i>
-------------	-----------------------------------

---

**Description**

Read bits from a bitstream

**Usage**

```
bs_read_bit(bs, n)
```

**Arguments**

bs	Bistream connection object created with bs_open()
n	number of bits to read



**Value**

logical vector of bit values

**Examples**

```
raw_vec <- as.raw(1:3)
bs <- bs_open(raw_vec, 'r')
bs_read_bit(bs, 4)
bs_is_aligned(bs)
bs_close(bs)
```

---

*bs\_write\_bit*                      *Write unaligned bits to a bitstream*

---

**Description**

Write unaligned bits to a bitstream

**Usage**

```
bs_write_bit(bs, x)
```

**Arguments**

- bs*                      Bistream connection object created with *bs\_open()*
- x*                        Logical vector of bit values

**Value**

Bitstream connection returned invisibly

**Examples**

```
bs <- bs_open(raw(), 'w')
bs_write_bit(bs, c(TRUE, FALSE, TRUE))
bs_align(bs, nbits = 8)
bs_flush(bs)
output <- bs_close(bs)
output
```

---

bs_write_byte	<i>Read/Write unaligned bytes with a bitstream</i>
---------------	--

---

### Description

Read/Write unaligned bytes with a bitstream

### Usage

```
bs_write_byte(bs, x)
```

```
bs_read_byte(bs, n)
```

### Arguments

bs	Bistream connection object created with <code>bs_open()</code>
x	vector of bytes to write. Integer vectors will be truncated to 8 bits before output. Numeric vectors will be rounded to integers and then truncated to 8 bits.
n	number of bytes to read

### Value

Reading returns a logical vector of bit values. When writing, the bs bitstream connection is returned invisibly

### Examples

```
bs <- bs_open(raw(), 'w')
bs_write_bit(bs, c(TRUE, FALSE))
bs_write_byte(bs, c(1, 2, 3))
bs_align(bs)
raw_vec <- bs_close(bs)

bs <- bs_open(raw_vec, 'r')
bs_read_bit(bs, 2)
bs_read_byte(bs, 3)
bs_close(bs)
```

---

`bs_write_sint_exp_golomb`*Read/Write Exponential-Golomb encoded signed integers*

---

**Description**

Read/Write Exponential-Golomb encoded signed integers

**Usage**

```
bs_write_sint_exp_golomb(bs, x)
```

```
bs_read_sint_exp_golomb(bs, n = 1L)
```

**Arguments**

<code>bs</code>	Bistream connection object created with <code>bs_open()</code>
<code>x</code>	integer vector to write
<code>n</code>	number of encoded integers to read

**Value**

Reading returns a vector of integers. Writing returns the bitstream invisibly.

**Examples**

```
bs <- bs_open(raw(), 'w')
bs_write_sint_exp_golomb(bs, c(0, 4, -21))
raw_vec <- bs_close(bs)
raw_vec
```

```
bs <- bs_open(raw_vec, 'r')
bs_read_sint_exp_golomb(bs, 3)
bs_close(bs)
```

---

`bs_write_uint`*Read/Write unsigned integers*

---

**Description**

Read/Write unsigned integers

**Usage**

```
bs_write_uint(bs, x, nbits)

bs_read_uint(bs, nbits, n = 1L)
```

**Arguments**

bs	Bistream connection object created with <code>bs_open()</code>
x	integer vector to write
nbits	the number of bits used for each integer
n	number of integers to read

**Value**

Reading returns a vector of non-negative integers. Writing returns the bitstream invisibly.

**Examples**

```
bs <- bs_open(raw(), 'w')
bs_write_uint(bs, c(0, 4, 21), nbits = 5)
bs_align(bs, 8)
raw_vec <- bs_close(bs)
raw_vec

bs <- bs_open(raw_vec, 'r')
bs_read_uint(bs, n = 3, nbits = 5)
bs_close(bs)
```

---

```
bs_write_uint_exp_golomb
```

*Read/Write Exponential-Golomb encoded non-negative integers*

---

**Description**

Read/Write Exponential-Golomb encoded non-negative integers

**Usage**

```
bs_write_uint_exp_golomb(bs, x)

bs_read_uint_exp_golomb(bs, n = 1L)
```

**Arguments**

bs	Bistream connection object created with <code>bs_open()</code>
x	integer vector to write
n	number of encoded integers to read

**Value**

Reading returns a vector of non-negative integers. Writing returns the bitstream invisibly.

**Examples**

```
bs <- bs_open(raw(), 'w')
bs_write_uint_exp_golomb(bs, c(0, 4, 21))
bs_align(bs, 8)
raw_vec <- bs_close(bs)
raw_vec
```

```
bs <- bs_open(raw_vec, 'r')
bs_read_uint_exp_golomb(bs, 3)
bs_close(bs)
```

---

is_bs	<i>Test if an object is a bitstream object</i>
-------	--

---

**Description**

Test if an object is a bitstream object

**Usage**

```
is_bs(x)
```

**Arguments**

x                    object to test

**Value**

logical. TRUE if object is a bitstream object

**Examples**

```
# Negative case
is_bs(NULL)

# Positive case
raw_vec <- as.raw(1:3)
bs <- bs_open(raw_vec, 'r')
is_bs(bs)
bs_close(bs)
```

---

pad_bits	<i>Pad a logical vector to the given size</i>
----------	---

---

**Description**

Pad a logical vector to the given size

**Usage**

```
pad_bits(bits, nbits = 8L, side = "left", value = FALSE)
```

**Arguments**

bits	logical vector
nbits	Pad width to a multiple of this number of bits
side	'left' or 'right'. Only the lowercase version of the first letter is used to determine the side. all of these are valid options: 'L', 'R', 'left', 'Right'
value	The value to use for padding. single logical value. Default: FALSE

**Value**

Logical vector with the specified padding

**Examples**

```
pad_bits(c(TRUE, TRUE), nbits = 8, side = 'left')
pad_bits(c(TRUE, TRUE), nbits = 8, side = 'R')
```

---

sint_to_exp_golomb_bits	<i>Convert between signed integers and Exponential-Golomb bit sequences</i>
-------------------------	---

---

**Description**

Convert between signed integers and Exponential-Golomb bit sequences

**Usage**

```
sint_to_exp_golomb_bits(x)

exp_golomb_bits_to_sint(bits, n = 1)
```

**Arguments**

x	integer vector with all values $\geq 0$
bits	logical vector of bit values
n	number of values to decode. Default: 1. Set to 'Inf' to decode all bits. Will raise an error if there are extra bits at the end that are unused.

**Value**

logical vector of bit values, or vector of signed integers

**Examples**

```
bits <- sint_to_exp_golomb_bits(c(0, 4, -21))
bits
exp_golomb_bits_to_sint(bits, n = 3)
```

---

uint\_to\_exp\_golomb\_bits

*Convert between non-negative integers and Exponential Golomb bit sequences*

---

**Description**

Convert between non-negative integers and Exponential Golomb bit sequences

**Usage**

```
uint_to_exp_golomb_bits(x)

exp_golomb_bits_to_uint(bits, n = 1)
```

**Arguments**

x	integer vector with all values $\geq 0$
bits	logical vector of bit values
n	number of values to decode. Default: 1. Set to 'Inf' to decode all bits. Will raise an error if there are extra bits at the end that are not properly encoded integers

**Value**

logical vector of bit values, or vector of non-negative integers

**Examples**

```
bits <- uint_to_exp_golomb_bits(c(0, 4, 21))
bits
exp_golomb_bits_to_uint(bits, n = 3)
```

# Index

assert\_bs, 2

bits\_to\_raw, 3  
bits\_to\_uint, 3  
bs\_advance, 4  
bs\_align, 5  
bs\_close (bs\_open), 7  
bs\_flush, 5  
bs\_is\_aligned, 6  
bs\_open, 7  
bs\_peek, 8  
bs\_read\_bit, 8  
bs\_read\_byte (bs\_write\_byte), 10  
bs\_read\_sint\_exp\_golomb  
    (bs\_write\_sint\_exp\_golomb), 11  
bs\_read\_uint (bs\_write\_uint), 11  
bs\_read\_uint\_exp\_golomb  
    (bs\_write\_uint\_exp\_golomb), 12  
bs\_write\_bit, 9  
bs\_write\_byte, 10  
bs\_write\_sint\_exp\_golomb, 11  
bs\_write\_uint, 11  
bs\_write\_uint\_exp\_golomb, 12

exp\_golomb\_bits\_to\_sint  
    (sint\_to\_exp\_golomb\_bits), 14  
exp\_golomb\_bits\_to\_uint  
    (uint\_to\_exp\_golomb\_bits), 15

is\_bs, 13

pad\_bits, 14

raw\_to\_bits (bits\_to\_raw), 3

sint\_to\_exp\_golomb\_bits, 14

uint\_to\_bits (bits\_to\_uint), 3  
uint\_to\_exp\_golomb\_bits, 15