



*Full credit is given to the above companies including the OS that this PDF file was generated!*

### ***Rocky Enterprise Linux 9.2 Manual Pages on command 'BIO\_s\_mem.3ossl'***

***\$ man BIO\_s\_mem.3ossl***

BIO\_S\_MEM(3ossl)            OpenSSL            BIO\_S\_MEM(3ossl)

#### NAME

BIO\_s\_secmem, BIO\_s\_mem, BIO\_set\_mem\_eof\_return, BIO\_get\_mem\_data,  
BIO\_set\_mem\_buf, BIO\_get\_mem\_ptr, BIO\_new\_mem\_buf - memory BIO

#### SYNOPSIS

```
#include <openssl/bio.h>
```

```
const BIO_METHOD *BIO_s_mem(void);
```

```
const BIO_METHOD *BIO_s_secmem(void);
```

```
BIO_set_mem_eof_return(BIO *b, int v);
```

```
long BIO_get_mem_data(BIO *b, char **pp);
```

```
BIO_set_mem_buf(BIO *b, BUF_MEM *bm, int c);
```

```
BIO_get_mem_ptr(BIO *b, BUF_MEM **pp);
```

```
BIO *BIO_new_mem_buf(const void *buf, int len);
```

## DESCRIPTION

`BIO_s_mem()` returns the memory BIO method function.

A memory BIO is a source/sink BIO which uses memory for its I/O. Data written to a memory BIO is stored in a `BUF_MEM` structure which is extended as appropriate to accommodate the stored data.

`BIO_s_secmem()` is like `BIO_s_mem()` except that the secure heap is used for buffer storage.

Any data written to a memory BIO can be recalled by reading from it. Unless the memory BIO is read only any data read from it is deleted from the BIO.

Memory BIOs support `BIO_gets()` and `BIO_puts()`.

If the `BIO_CLOSE` flag is set when a memory BIO is freed then the underlying `BUF_MEM` structure is also freed.

Calling `BIO_reset()` on a read write memory BIO clears any data in it if the flag `BIO_FLAGS_NONCLEAR_RST` is not set, otherwise it just restores the read pointer to the state it was just after the last write was performed and the data can be read again. On a read only BIO it similarly restores the BIO to its original state and the read only data can be read again.

`BIO_eof()` is true if no data is in the BIO.

`BIO_ctrl_pending()` returns the number of bytes currently stored.

`BIO_set_mem_eof_return()` sets the behaviour of memory BIO `b` when it is empty. If the `v` is zero then an empty memory BIO will return EOF (that

is it will return zero and `BIO_should_retry(b)` will be false. If `v` is non zero then it will return `v` when it is empty and it will set the read retry flag (that is `BIO_read_retry(b)` is true). To avoid ambiguity with a normal positive return value `v` should be set to a negative value, typically `-1`.

`BIO_get_mem_data()` sets `*pp` to a pointer to the start of the memory BIOs data and returns the total amount of data available. It is implemented as a macro.

`BIO_set_mem_buf()` sets the internal `BUF_MEM` structure to `bm` and sets the close flag to `c`, that is `c` should be either `BIO_CLOSE` or `BIO_NOCLOSE`. It is a macro.

`BIO_get_mem_ptr()` places the underlying `BUF_MEM` structure in `*pp`. It is a macro.

`BIO_new_mem_buf()` creates a memory BIO using `len` bytes of data at `buf`, if `len` is `-1` then the `buf` is assumed to be nul terminated and its length is determined by `strlen`. The BIO is set to a read only state and as a result cannot be written to. This is useful when some data needs to be made available from a static area of memory in the form of a BIO. The supplied data is read directly from the supplied buffer: it is not copied first, so the supplied area of memory must be unchanged until the BIO is freed.

## NOTES

Writes to memory BIOs will always succeed if memory is available: that is their size can grow indefinitely.

Every write after partial read (not all data in the memory buffer was read) to a read write memory BIO will have to move the unread data with an internal copy operation, if a BIO contains a lot of data and it is

read in small chunks intertwined with writes the operation can be very slow. Adding a buffering BIO to the chain can speed up the process.

Calling `BIO_set_mem_buf()` on a BIO created with `BIO_new_secmem()` will give undefined results, including perhaps a program crash.

Switching the memory BIO from read write to read only is not supported and can give undefined results including a program crash. There are two notable exceptions to the rule. The first one is to assign a static memory buffer immediately after BIO creation and set the BIO as read only.

The other supported sequence is to start with read write BIO then temporarily switch it to read only and call `BIO_reset()` on the read only BIO immediately before switching it back to read write. Before the BIO is freed it must be switched back to the read write mode.

Calling `BIO_get_mem_ptr()` on read only BIO will return a `BUF_MEM` that contains only the remaining data to be read. If the close status of the BIO is set to `BIO_NOCLOSE`, before freeing the `BUF_MEM` the data pointer in it must be set to `NULL` as the data pointer does not point to an allocated memory.

Calling `BIO_reset()` on a read write memory BIO with `BIO_FLAGS_NONCLEAR_RST` flag set can have unexpected outcome when the reads and writes to the BIO are intertwined. As documented above the BIO will be reset to the state after the last completed write operation. The effects of reads preceding that write operation cannot be undone.

Calling `BIO_get_mem_ptr()` prior to a `BIO_reset()` call with `BIO_FLAGS_NONCLEAR_RST` set has the same effect as a write operation.

## BUGS

There should be an option to set the maximum size of a memory BIO.

## RETURN VALUES

BIO\_s\_mem() and BIO\_s\_secmem() return a valid memory BIO\_METHOD structure.

BIO\_set\_mem\_eof\_return(), BIO\_set\_mem\_buf() and BIO\_get\_mem\_ptr() return 1 on success or a value which is less than or equal to 0 if an error occurred.

BIO\_get\_mem\_data() returns the total number of bytes available on success, 0 if b is NULL, or a negative value in case of other errors.

BIO\_new\_mem\_buf() returns a valid BIO structure on success or NULL on error.

## EXAMPLES

Create a memory BIO and write some data to it:

```
BIO *mem = BIO_new(BIO_s_mem());
```

```
BIO_puts(mem, "Hello World\n");
```

Create a read only memory BIO:

```
char data[] = "Hello World";
```

```
BIO *mem = BIO_new_mem_buf(data, -1);
```

Extract the BUF\_MEM structure from a memory BIO and then free up the BIO:

```
BUF_MEM *bptr;
```

```
BIO_get_mem_ptr(mem, &bptr);  
BIO_set_close(mem, BIO_NOCLOSE); /* So BIO_free() leaves BUF_MEM alone */  
BIO_free(mem);
```

## COPYRIGHT

Copyright 2000-2020 The OpenSSL Project Authors. All Rights Reserved.

Licensed under the Apache License 2.0 (the "License"). You may not use this file except in compliance with the License. You can obtain a copy in the file LICENSE in the source distribution or at <https://www.openssl.org/source/license.html>.

3.0.7                    2023-07-13                    BIO\_S\_MEM(3ossl)