NAME

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>

BIO_METHOD * BIO_s_mem(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(void *buf, int len);
```

DESCRIPTION

BIO_s_mem() return 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.

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. On a read only BIO it 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 null 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 read from a read write memory BIO will remove the data just read with an internal copy operation, if a BIO contains a lot of data and it is read in small chunks the operation can be very slow. The use of a read only memory BIO avoids this problem. If the BIO must be read write then adding a buffering BIO to the chain will speed up the process.

BUGS

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

There should be a way to ‘‘rewind’’ a read write BIO without destroying its contents.

The copying operation should not occur after every small read of a large BIO to improve efficiency.

EXAMPLE

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;
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);
```

SEE ALSO

TBA