NAME

BIO_read_ex, BIO_write_ex, BIO_read, BIO_write, BIO_gets, BIO_puts – BIO I/O functions

SYNOPSIS

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

int BIO_read_ex(BIO *b, void *data, size_t dlen, size_t *readbytes);
int BIO_write_ex(BIO *b, const void *data, size_t dlen, size_t *written);

int BIO_read(BIO *b, void *data, int dlen);
int BIO_gets(BIO *b, char *buf, int size);
int BIO_write(BIO *b, const void *data, int dlen);
int BIO_puts(BIO *b, const char *buf);
```

DESCRIPTION

BIO_read_ex() attempts to read **dlen** bytes from BIO **b** and places the data in **data**. If any bytes were successfully read then the number of bytes read is stored in ***readbytes**.

BIO_write_ex() attempts to write **dlen** bytes from **data** to BIO **b**. If successful then the number of bytes written is stored in ***written**.

BIO_read() attempts to read **len** bytes from BIO **b** and places the data in **buf**.

BIO_gets() performs the BIOs "gets" operation and places the data in **buf**. Usually this operation will attempt to read a line of data from the BIO of maximum length **size−1**. There are exceptions to this, however; for example, **BIO_gets()** on a digest BIO will calculate and return the digest and other BIOs may not support **BIO_gets()** at all. The returned string is always NUL-terminated and the '\n' is preserved if present in the input data.

BIO_write() attempts to write **len** bytes from **buf** to BIO **b**.

BIO_puts() attempts to write a NUL-terminated string **buf** to BIO **b**.

RETURN VALUES

BIO_read_ex() and **BIO_write_ex()** return 1 if data was successfully read or written, and 0 otherwise.

All other functions return either the amount of data successfully read or written (if the return value is positive) or that no data was successfully read or written if the result is 0 or −1. If the return value is −2 then the operation is not implemented in the specific BIO type. The trailing NUL is not included in the length returned by **BIO_gets()**.

NOTES

A 0 or −1 return is not necessarily an indication of an error. In particular when the source/sink is nonblocking or of a certain type it may merely be an indication that no data is currently available and that the application should retry the operation later.

One technique sometimes used with blocking sockets is to use a system call (such as **select()**, **poll()** or equivalent) to determine when data is available and then call **read()** to read the data. The equivalent with BIOs (that is call **select()** on the underlying I/O structure and then call **BIO_read()** to read the data) should **not** be used because a single call to **BIO_read()** can cause several reads (and writes in the case of SSL BIOs) on the underlying I/O structure and may block as a result. Instead **select()** (or equivalent) should be combined with non blocking I/O so successive reads will request a retry instead of blocking.

See **BIO_should_retry(3)** for details of how to determine the cause of a retry and other I/O issues.

If the **BIO_gets()** function is not supported by a BIO then it possible to work around this by adding a buffering BIO **BIO_f_buffer(3)** to the chain.

SEE ALSO

BIO_should_retry(3)

HISTORY

BIO_gets() on 1.1.0 and older when called on **BIO_fd()** based BIO does not keep the '\n' at the end of the line in the buffer.

COPYRIGHT