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

Red Hat Enterprise Linux Release 9.2 Manual Pages on 'BIO_write_ex.3ossl' command

$ man BIO_write_ex.3ossl

BIO_READ(3ossl) OpenSSL BIO_READ(3ossl)

NAME

BIO_read_ex, BIO_write_ex, BIO_read, BIO_write, BIO_gets, BIO_get_line,

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_get_line(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 unless written is NULL.

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. On binary input there may be NUL characters within the string; in this case the return value (if nonnegative) may give an incorrect length.

BIO_get_line() attempts to read from BIO <b> a line of data up to the next '\n' or the maximum length size-1 is reached and places the data in buf. The returned string is always NUL-terminated and the '\n' is preserved if present in the input data. On binary input there may be NUL characters within the string; in this case the return value (if nonnegative) gives the actual length read. For implementing this, unfortunately the data needs to be read byte-by-byte.

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() returns 1 if data was successfully read, and 0 otherwise.

BIO_write_ex() returns 1 if no error was encountered writing data, 0

otherwise. Requesting to write 0 bytes is not considered an error.

BIO_write() returns -2 if the "write" operation is not implemented by the BIO or -1 on other errors. Otherwise it returns the number of bytes written. This may be 0 if the BIO b is NULL or dlen <= 0.

BIO_gets() returns -2 if the "gets" operation is not implemented by the BIO or -1 on other errors. Otherwise it typically returns the amount of data read, but depending on the implementation it may return only the length up to the first NUL character contained in the data read. In any case the trailing NUL that is added after the data read is not included in the length returned.

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.

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 "gets" method is not supported by a BIO then BIO_get_line() can be used. It is also possible to make BIO_gets() usable even if the "gets" method is not supported 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 did not keep the '\n' at the end of the line in the buffer.

BIO_get_line() was added in OpenSSL 3.0.

BIO_write_ex() returns 1 if the size of the data to write is 0 and the written parameter of the function can be NULL since OpenSSL 3.0.

COPYRIGHT