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_should_read.3ossl' command

$ man BIO_should_read.3ossl

BIO_SHOULD_RETRY(3ossl) OpenSSL BIO_SHOULD_RETRY(3ossl)

NAME

BIO_should_read, BIO_should_write, BIO_should_io_special,

BIO_retry_type, BIO_should_retry, BIO_get_retry_BIO,

BIO_get_retry_reason, BIO_set_retry_reason - BIO retry functions

SYNOPSIS

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

int BIO_should_read(BIO *b);

int BIO_should_write(BIO *b);

int BIO_should_io_special(iBIO *b);

int BIO_retry_type(BIO *b);

int BIO_should_retry(BIO *b);

BIO *BIO_get_retry_BIO(BIO *bio, int *reason);

int BIO_get_retry_reason(BIO *bio);

void BIO_set_retry_reason(BIO *bio, int reason);
```

DESCRIPTION

These functions determine why a BIO is not able to read or write data.

They will typically be called after a failed BIO_read_ex() or

BIO_write_ex() call.

BIO_should_retry() is true if the call that produced this condition

should then be retried at a later time.

If BIO_should_retry() is false then the cause is an error condition.

BIO_should_read() is true if the cause of the condition is that the BIO

has insufficient data to return. Check for readability and/or retry the last operation.

BIO_should_write() is true if the cause of the condition is that the BIO has pending data to write. Check for writability and/or retry the last operation.

BIO_should_io_special() is true if some "special" condition, that is a reason other than reading or writing is the cause of the condition.

BIO_retry_type() returns a mask of the cause of a retry condition consisting of the values BIO_FLAGS_READ, BIO_FLAGS_WRITE, BIO_FLAGS_IO_SPECIAL though current BIO types will only set one of these.

BIO_get_retry_BIO() determines the precise reason for the special condition, it returns the BIO that caused this condition and if reason is not NULL it contains the reason code. The meaning of the reason code and the action that should be taken depends on the type of BIO that resulted in this condition.

BIO_get_retry_reason() returns the reason for a special condition if passed the relevant BIO, for example as returned by BIO_get_retry_BIO().

BIO_set_retry_reason() sets the retry reason for a special condition for a given BIO. This would usually only be called by BIO implementations.

NOTES

BIO_should_read(), BIO_should_write(), BIO_should_io_special(), BIO_retry_type(), and BIO_should_retry(), are implemented as macros.

If BIO_should_retry() returns false then the precise "error condition" depends on the BIO type that caused it and the return code of the BIO operation. For example if a call to BIO_read_ex() on a socket BIO returns 0 and BIO_should_retry() is false then the cause will be that the connection closed. A similar condition on a file BIO will mean that it has reached EOF. Some BIO types may place additional information on the error queue. For more details see the individual BIO type manual pages.

If the underlying I/O structure is in a blocking mode almost all current BIO types will not request a retry, because the underlying I/O calls will not. If the application knows that the BIO type will never signal a retry then it need not call BIO_should_retry() after a failed BIO I/O call. This is typically done with file BIOs.

SSL BIOs are the only current exception to this rule: they can request a retry even if the underlying I/O structure is blocking, if a handshake occurs during a call to BIO_read(). An application can retry the failed call immediately or avoid this situation by setting SSL_MODE_AUTO_RETRY on the underlying SSL structure.

While an application may retry a failed non blocking call immediately this is likely to be very inefficient because the call will fail repeatedly until data can be processed or is available. An application will normally wait until the necessary condition is satisfied. How this is done depends on the underlying I/O structure.

For example if the cause is ultimately a socket and BIO_should_read() is true then a call to select() may be made to wait until data is available and then retry the BIO operation. By combining the retry conditions of several non blocking BIOs in a single select() call it is possible to service several BIOs in a single thread, though the performance may be poor if SSL BIOs are present because long delays can occur during the initial handshake process.

It is possible for a BIO to block indefinitely if the underlying I/O structure cannot process or return any data. This depends on the behaviour of the platforms I/O functions. This is often not desirable: one solution is to use non blocking I/O and use a timeout on the select() (or equivalent) call.

BUGS

The OpenSSL ASN1 functions cannot gracefully deal with non blocking I/O: that is they cannot retry after a partial read or write. This is usually worked around by only passing the relevant data to ASN1 functions when the entire structure can be read or written.

RETURN VALUES

BIO_should_read(), BIO_should_write(), BIO_should_io_special(), and BIO_should_retry() return either 1 or 0 based on the actual conditions of the BIO.

BIO_retry_type() returns a flag combination presenting the cause of a retry condition or false if there is no retry condition.

BIO_get_retry_BIO() returns a valid BIO structure.

BIO_get_retry_reason() returns the reason for a special condition.

SEE ALSO

bio(7)

HISTORY

The BIO_get_retry_reason() and BIO_set_retry_reason() functions were added in OpenSSL 1.1.0.

COPYRIGHT