NAME

BIO_s_dgram_pair, BIO_new_bio_dgram_pair, BIO_dgram_set_no_trunc, BIO_dgram_get_no_trunc, BIO_dgram_get_effective_caps, BIO_dgram_get_caps, BIO_dgram_set_caps, BIO_dgram_set_mtu, BIO_dgram_get_mtu, BIO_dgram_set0_local_addr − datagram pair BIO

SYNOPSIS

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

const BIO_METHOD *BIO_s_dgram_pair(void);

int BIO_new_bio_dgram_pair(BIO **bio1, size_t writebuf1,
                           BIO **bio2, size_t writebuf2);
int BIO_dgram_set_no_trunc(BIO *bio, int enable);
int BIO_dgram_get_no_trunc(BIO *bio);
uint32_t BIO_dgram_get_effective_caps(BIO *bio);
uint32_t BIO_dgram_get_caps(BIO *bio);
int BIO_dgram_set_caps(BIO *bio, uint32_t caps);
int BIO_dgram_set_mtu(BIO *bio, unsigned int mtu);
unsigned int BIO_dgram_get_mtu(BIO *bio);
int BIO_dgram_set0_local_addr(BIO *bio, BIO_ADDR *addr);
```

DESCRIPTION

BIO_s_dgram_pair() returns the method for a BIO datagram pair. A BIO datagram pair is similar to a BIO pair (see **BIO_s_bio**(3)) but has datagram semantics. Broadly, this means that the length of the buffer passed to a write call will match that retrieved by a read call. If the buffer passed to a read call is too short, the datagram is truncated or the read fails, depending on how the BIO is configured.

The BIO datagram pair attaches certain metadata to each write, such as source and destination addresses. This information may be retrieved on read.

A typical application of a BIO datagram pair is to allow an application to keep all datagram network I/O requested by libssl under application control.

The BIO datagram pair is designed to support multithreaded use where certain restrictions are observed; see THREADING.

The BIO datagram pair allows each half of a pair to signal to the other half whether they support certain capabilities; see CAPABILITY INDICATION.

BIO_new_bio_dgram_pair() combines the calls to **BIO_new**(3), **BIO_make_bio_pair**(3) and **BIO_set_write_buf_size**(3) to create a connected pair of BIOs **bio1**, **bio2** with write buffer sizes **writebuf1** and **writebuf2**. If either size is zero then the default size is used.

BIO_make_bio_pair(3) may be used to join two datagram pair BIOs into a pair. The two BIOs must both use the method returned by **BIO_s_dgram_pair()** and neither of the BIOs may currently be associated in a pair.

BIO_destroy_bio_pair(3) destroys the association between two connected BIOs. Freeing either half of the pair will automatically destroy the association.

BIO_reset(3) clears any data in the write buffer of the given BIO. This means that the opposite BIO in the pair will no longer have any data waiting to be read.

The BIO maintains a fixed size internal write buffer. When the buffer is full, further writes will fail until the buffer is drained via calls to **BIO_read**(3). The size of the buffer can be changed using **BIO_set_write_buf_size**(3) and queried using **BIO_get_write_buf_size**(3).

Note that the write buffer is partially consumed by metadata stored internally which is attached to each datagram, such as source and destination addresses. The size of this overhead is undefined and may change between releases.

The standard **BIO_ctrl_pending**(3) call has modified behaviour and returns the size of the next datagram

waiting to be read in bytes. An application can use this function to ensure it provides an adequate buffer to a subsequent read call. If no datagram is waiting to be read, zero is returned.

This BIO does not support sending or receiving zero−length datagrams. Passing a zero−length buffer to BIO_write is treated as a no−op.

BIO_eof (3) returns 1 only if the given BIO datagram pair BIO is not currently connected to a peer BIO.

BIO_get_write_guarantee (3) and **BIO_ctrl_get_write_guarantee** (3) return how large a datagram the next call to **BIO_write** (3) can accept. If there is not enough space in the write buffer to accept another datagram equal in size to the configured MTU, zero is returned (see below). This is intended to avoid a situation where an application attempts to read a datagram from a network intending to write it to a BIO datagram pair, but where the received datagram ends up being too large to write to the BIO datagram pair.

BIO_dgram_set_no_trunc() and **BIO_ctrl_get_no_trunc()** set and retrieve the truncation mode for the given half of a BIO datagram pair. When no−truncate mode is enabled, **BIO_read()** will fail if the buffer provided is inadequate to hold the next datagram to be read. If no−truncate mode is disabled (the default), the datagram will be silently truncated. This default behaviour maintains compatibility with the semantics of the Berkeley sockets API.

BIO_dgram_set_mtu() and **BIO_dgram_get_mtu()** may be used to set an informational MTU value on the BIO datagram pair. If **BIO_dgram_set_mtu()** is used on a BIO which is currently part of a BIO datagram pair, the MTU value is set on both halves of the pair. The value does not affect the operation of the BIO datagram pair (except for **BIO_get_write_guarantee()**; see above) but may be used by other code to determine a requested MTU. When a BIO datagram pair BIO is created, the MTU is set to an unspecified but valid value.

BIO_dgram_set0_local_addr() can be used to set the local BIO_ADDR to be used when sending a datagram via a BIO datagram pair. This becomes the peer address when receiving on the other half of the pair. If the BIO is used in a call to **BIO_sendmmsg** (3) and a local address is explicitly specified, then the explicitly specified local address takes precedence. The reference to the BIO_ADDR is passed to the BIO by this call and will be freed automatically when the BIO is freed.

BIO_flush (3) is a no−op.

NOTES

The halves of a BIO datagram pair have independent lifetimes and must be separately freed.

THREADING

BIO_recvmmsg (3), **BIO_sendmmsg** (3), **BIO_read** (3), **BIO_write** (3), **BIO_pending** (3), **BIO_get_write_guarantee** (3) and **BIO_flush** (3) may be used by multiple threads simultaneously on the same BIO datagram pair. Specific **BIO_ctrl** (3) operations (namely BIO_CTRL_PENDING, BIO_CTRL_FLUSH and BIO_C_GET_WRITE_GUARANTEE) may also be used. Invoking any other BIO call, or any other **BIO_ctrl** (3) operation, on either half of a BIO datagram pair while any other BIO call is also in progress to either half of the same BIO datagram pair results in undefined behaviour.

CAPABILITY INDICATION

The BIO datagram pair can be used to enqueue datagrams which have source and destination addresses attached. It is important that the component consuming one side of a BIO datagram pair understand whether the other side of the pair will honour any source and destination addresses it attaches to each datagram. For example, if datagrams are queued with destination addresses set but simply read by simple calls to **BIO_read** (3), the destination addresses will be discarded.

Each half of a BIO datagram pair can have capability flags set on it which indicate whether source and destination addresses will be honoured by the reader and whether they will be provided by the writer. These capability flags should be set via a call to **BIO_dgram_set_caps()**, and these capabilities will be reflected in the value returned by **BIO_dgram_get_effective_caps()** on the opposite BIO. If necessary, the capability value previously set can be retrieved using **BIO_dgram_get_caps()**. Note that **BIO_dgram_set_caps()** on a given BIO controls the capabilities advertised to the peer, and **BIO_dgram_get_effective_caps()** on a given BIO determines the capabilities advertised by the peer of that BIO.

The following capabilities are available:

BIO_DGRAM_CAP_HANDLES_SRC_ADDR
: The user of the datagram pair BIO promises to honour source addresses provided with datagrams written to the BIO pair.

BIO_DGRAM_CAP_HANDLES_DST_ADDR
: The user of the datagram pair BIO promises to honour destination addresses provided with datagrams written to the BIO pair.

BIO_DGRAM_CAP_PROVIDES_SRC_ADDR
: The user of the datagram pair BIO advertises the fact that it will provide source addressing information with future writes to the BIO pair, where available.

BIO_DGRAM_CAP_PROVIDES_DST_ADDR
: The user of the datagram pair BIO advertises the fact that it will provide destination addressing information with future writes to the BIO pair, where available.

If a caller attempts to specify a destination address (for example, using **BIO_sendmmsg**(3)) and the peer has not advertised the **BIO_DGRAM_CAP_HANDLES_DST_ADDR** capability, the operation fails. Thus, capability negotiation is mandatory.

If a caller attempts to specify a source address when writing, or requests a destination address when receiving, and local address support has not been enabled, the operation fails; see **BIO_dgram_set_local_addr_enable**(3).

If a caller attempts to enable local address support using **BIO_dgram_set_local_addr_enable**(3) and **BIO_dgram_get_local_addr_cap**(3) does not return 1 (meaning that the peer has not advertised both the **BIO_DGRAM_CAP_HANDLES_SRC_ADDR** and the **BIO_DGRAM_CAP_PROVIDES_DST_ADDR** capability), the operation fails.

BIO_DGRAM_CAP_PROVIDES_SRC_ADDR and **BIO_DGRAM_CAP_PROVIDES_DST_ADDR** indicate that the application using that half of a BIO datagram pair promises to provide source and destination addresses respectively when writing datagrams to that half of the BIO datagram pair. However, these capability flags do not affect the behaviour of the BIO datagram pair.

RETURN VALUES

BIO_new_bio_dgram_pair() returns 1 on success, with the new BIOs available in **bio1** and **bio2**, or 0 on failure, with NULL pointers stored into the locations for **bio1** and **bio2**. Check the error stack for more information.

BIO_dgram_set_no_trunc(), **BIO_dgram_set_caps()** and **BIO_dgram_set_mtu()** return 1 on success and 0 on failure.

BIO_dgram_get_no_trunc() returns 1 if no−truncate mode is enabled on a BIO, or 0 if no−truncate mode is not enabled or not supported on a given BIO.

BIO_dgram_get_effective_caps() and **BIO_dgram_get_caps()** return zero if no capabilities are supported.

BIO_dgram_get_mtu() returns the MTU value configured on the BIO, or zero if the operation is not supported.

BIO_dgram_set0_local_addr() returns 1 on success and <= 0 otherwise.

SEE ALSO

BIO_s_bio(3), **bio**(7)

HISTORY

BIO_s_dgram_pair(), **BIO_new_bio_dgram_pair()** were added in OpenSSL 3.2.

COPYRIGHT