NAME

BIO_s_accept, BIO_set_accept_name, BIO_set_accept_port, BIO_get_accept_name, BIO_get_accept_port, BIO_new_accept, BIO_set_nbio_accept, BIO_set_accept_bios, BIO_get_peer_name, BIO_get_peer_port, BIO_get_accept_ip_family, BIO_set_accept_ip_family, BIO_set_bind_mode, BIO_get_bind_mode, BIO_do_accept - accept BIO

SYNOPSIS

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

const BIO_METHOD *BIO_s_accept(void);

long BIO_set_accept_name(BIO *b, char *name);
char *BIO_get_accept_name(BIO *b);

long BIO_set_accept_port(BIO *b, char *port);
char *BIO_get_accept_port(BIO *b);

BIO *BIO_new_accept(char *host_port);

long BIO_set_nbio_accept(BIO *b, int n);
long BIO_set_accept_bios(BIO *b, char *bio);

char *BIO_get_peer_name(BIO *b);
char *BIO_get_peer_port(BIO *b);
long BIO_get_accept_ip_family(BIO *b);
long BIO_set_accept_ip_family(BIO *b, long family);

long BIO_set_bind_mode(BIO *b, long mode);
long BIO_get_bind_mode(BIO *b);

int BIO_do_accept(BIO *b);
```

DESCRIPTION

BIO_s_accept() returns the accept BIO method. This is a wrapper round the platform's TCP/IP socket accept routines.

Using accept BIOs, TCP/IP connections can be accepted and data transferred using only BIO routines. In this way any platform specific operations are hidden by the BIO abstraction.

Read and write operations on an accept BIO will perform I/O on the underlying connection. If no

connection is established and the port (see below) is set up properly then the BIO waits for an incoming connection.

Accept BIOs support **BIO_puts()** but not **BIO_gets()**.

If the close flag is set on an accept BIO then any active connection on that chain is shutdown and the socket closed when the BIO is freed.

Calling **BIO_reset()** on an accept BIO will close any active connection and reset the BIO into a state where it awaits another incoming connection.

BIO_get_fd() and **BIO_set_fd()** can be called to retrieve or set the accept socket. See **BIO_s_fd**(3)

BIO_set_accept_name() uses the string **name** to set the accept name. The name is represented as a string of the form "host:port", where "host" is the interface to use and "port" is the port. The host can be "*" or empty which is interpreted as meaning any interface. If the host is an IPv6 address, it has to be enclosed in brackets, for example "[::1]:https". "port" has the same syntax as the port specified in **BIO_set_conn_port()** for connect BIOs, that is it can be a numerical port string or a string to lookup using **getservbyname()** and a string table.

BIO_set_accept_port() uses the string **port** to set the accept port of BIO *b*. "port" has the same syntax as the port specified in **BIO_set_conn_port()** for connect BIOs, that is it can be a numerical port string or a string to lookup using **getservbyname()** and a string table. If the given port is 0 then a random available port is chosen. It may be queried using **BIO_sock_info()** and **BIO_ADDR_service_string**(3).

BIO_new_accept() combines **BIO_new()** and **BIO_set_accept_name()** into a single call: that is it creates a new accept BIO with port **host_port**.

BIO_set_nbio_accept() sets the accept socket to blocking mode (the default) if **n** is 0 or non blocking mode if **n** is 1.

BIO_set_accept_bios() can be used to set a chain of BIOs which will be duplicated and prepended to the chain when an incoming connection is received. This is useful if, for example, a buffering or SSL BIO is required for each connection. The chain of BIOs must not be freed after this call, they will be automatically freed when the accept BIO is freed.

BIO_get_accept_ip_family() returns the IP family accepted by the BIO *b*, which may be **BIO_FAMILY_IPV4**, **BIO_FAMILY_IPV6**, or **BIO_FAMILY_IPANY**.

BIO_set_accept_ip_family() sets the IP family *family* accepted by BIO *b*. The default is

BIO_FAMILY_IPANY.

BIO_set_bind_mode() and **BIO_get_bind_mode()** set and retrieve the current bind mode. If **BIO_BIND_NORMAL** (the default) is set then another socket cannot be bound to the same port. If **BIO_BIND_REUSEADDR** is set then other sockets can bind to the same port. If **BIO_BIND_REUSEADDR_IF_UNUSED** is set then and attempt is first made to use BIO_BIN_NORMAL, if this fails and the port is not in use then a second attempt is made using **BIO_BIND_REUSEADDR**.

BIO_do_accept() serves two functions. When it is first called, after the accept BIO has been setup, it will attempt to create the accept socket and bind an address to it. Second and subsequent calls to **BIO_do_accept()** will await an incoming connection, or request a retry in non blocking mode.

NOTES

When an accept BIO is at the end of a chain it will await an incoming connection before processing I/O calls. When an accept BIO is not at then end of a chain it passes I/O calls to the next BIO in the chain.

When a connection is established a new socket BIO is created for the connection and appended to the chain. That is the chain is now accept->socket. This effectively means that attempting I/O on an initial accept socket will await an incoming connection then perform I/O on it.

If any additional BIOs have been set using **BIO_set_accept_bios()** then they are placed between the socket and the accept BIO, that is the chain will be accept->otherbios->socket.

If a server wishes to process multiple connections (as is normally the case) then the accept BIO must be made available for further incoming connections. This can be done by waiting for a connection and then calling:

```
connection = BIO_pop(accept);
```

After this call **connection** will contain a BIO for the recently established connection and **accept** will now be a single BIO again which can be used to await further incoming connections. If no further connections will be accepted the **accept** can be freed using **BIO_free()**.

If only a single connection will be processed it is possible to perform I/O using the accept BIO itself. This is often undesirable however because the accept BIO will still accept additional incoming connections. This can be resolved by using **BIO_pop()** (see above) and freeing up the accept BIO after the initial connection.

If the underlying accept socket is nonblocking and **BIO_do_accept()** is called to await an incoming

connection it is possible for **BIO_should_io_special()** with the reason BIO_RR_ACCEPT. If this happens then it is an indication that an accept attempt would block: the application should take appropriate action to wait until the underlying socket has accepted a connection and retry the call.

BIO_set_accept_name(), **BIO_get_accept_name()**, **BIO_set_accept_port()**, **BIO_get_accept_port()**, **BIO_set_nbio_accept()**, **BIO_set_accept_bios()**, **BIO_get_peer_name()**, **BIO_get_peer_port()**, **BIO_get_accept_ip_family()**, **BIO_set_accept_ip_family()**, **BIO_set_bind_mode()**, **BIO_get_bind_mode()** and **BIO_do_accept()** are macros.

RETURN VALUES

BIO_do_accept(), **BIO_set_accept_name()**, **BIO_set_accept_port()**, **BIO_set_nbio_accept()**, **BIO_set_accept_bios()**, **BIO_set_accept_ip_family()**, and **BIO_set_bind_mode()** return 1 for success and <=0 for failure.

BIO_get_accept_name() returns the accept name or NULL on error. **BIO_get_peer_name()** returns the peer name or NULL on error.

BIO_get_accept_port() returns the accept port as a string or NULL on error. **BIO_get_peer_port()** returns the peer port as a string or NULL on error. **BIO_get_accept_ip_family()** returns the IP family or <=0 on error.

BIO_get_bind_mode() returns the set of **BIO_BIND** flags, or <=0 on failure.

BIO_new_accept() returns a BIO or NULL on error.

EXAMPLES

This example accepts two connections on port 4444, sends messages down each and finally closes both down.

```
BIO *abio, *cbio, *cbio2;

/* First call to BIO_do_accept() sets up accept BIO */
abio = BIO_new_accept("4444");
if (BIO_do_accept(abio) <= 0) {
    fprintf(stderr, "Error setting up accept\n");
    ERR_print_errors_fp(stderr);
    exit(1);
}

/* Wait for incoming connection */
```

```
if (BIO_do_accept(abio) <= 0) {
    fprintf(stderr, "Error accepting connection\n");
    ERR_print_errors_fp(stderr);
    exit(1);
}
fprintf(stderr, "Connection 1 established\n");

/* Retrieve BIO for connection */
cbio = BIO_pop(abio);
BIO_puts(cbio, "Connection 1: Sending out Data on initial connection\n");
fprintf(stderr, "Sent out data on connection 1\n");

/* Wait for another connection */
if (BIO_do_accept(abio) <= 0) {
    fprintf(stderr, "Error accepting connection\n");
    ERR_print_errors_fp(stderr);
    exit(1);
}
fprintf(stderr, "Connection 2 established\n");

/* Close accept BIO to refuse further connections */
cbio2 = BIO_pop(abio);
BIO_free(abio);
BIO_puts(cbio2, "Connection 2: Sending out Data on second\n");
fprintf(stderr, "Sent out data on connection 2\n");

BIO_puts(cbio, "Connection 1: Second connection established\n");

/* Close the two established connections */
BIO_free(cbio);
BIO_free(cbio2);
```

COPYRIGHT