

**NAME**

BIO\_f\_base64 - base64 BIO filter

**SYNOPSIS**

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

BIO_METHOD * BIO_f_base64(void);
```

**DESCRIPTION**

*BIO\_f\_base64()* returns the base64 BIO method. This is a filter BIO that base64 encodes any data written through it and decodes any data read through it.

Base64 BIOs do not support *BIO\_gets()* or *BIO\_puts()*.

*BIO\_flush()* on a base64 BIO that is being written through is used to signal that no more data is to be encoded: this is used to flush the final block through the BIO.

The flag `BIO_FLAGS_BASE64_NO_NL` can be set with *BIO\_set\_flags()* to encode the data all on one line or expect the data to be all on one line.

**NOTES**

Because of the format of base64 encoding the end of the encoded block cannot always be reliably determined.

**RETURN VALUES**

*BIO\_f\_base64()* returns the base64 BIO method.

**EXAMPLES**

Base64 encode the string "Hello World\n" and write the result to standard output:

```
BIO *bio, *b64;
char message[] = "Hello World \n";

b64 = BIO_new(BIO_f_base64());
bio = BIO_new_fp(stdout, BIO_NOCLOSE);
BIO_push(b64, bio);
BIO_write(b64, message, strlen(message));
BIO_flush(b64);

BIO_free_all(b64);
```

Read Base64 encoded data from standard input and write the decoded data to standard output:

```
BIO *bio, *b64, *bio_out;
char inbuf[512];
int inlen;

b64 = BIO_new(BIO_f_base64());
bio = BIO_new_fp(stdin, BIO_NOCLOSE);
bio_out = BIO_new_fp(stdout, BIO_NOCLOSE);
BIO_push(b64, bio);
while((inlen = BIO_read(b64, inbuf, 512)) > 0)
    BIO_write(bio_out, inbuf, inlen);

BIO_flush(bio_out);
BIO_free_all(b64);
```

**BUGS**

The ambiguity of EOF in base64 encoded data can cause additional data following the base64 encoded block to be misinterpreted.

There should be some way of specifying a test that the BIO can perform to reliably determine EOF (for example a MIME boundary).

**SEE ALSO**

TBA