



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 'OSSL_HTTP_REQ_CTX_free.3ossl' command

```
$ man OSSL_HTTP_REQ_CTX_free.3ossl
```

```
OSSL_HTTP_REQ_CTX(3ossl)      OpenSSL      OSSL_HTTP_REQ_CTX(3ossl)
```

NAME

OSSL_HTTP_REQ_CTX, OSSL_HTTP_REQ_CTX_new, OSSL_HTTP_REQ_CTX_free, OSSL_HTTP_REQ_CTX_set_request_line, OSSL_HTTP_REQ_CTX_add1_header, OSSL_HTTP_REQ_CTX_set_expected, OSSL_HTTP_REQ_CTX_set1_req, OSSL_HTTP_REQ_CTX_nbio, OSSL_HTTP_REQ_CTX_nbio_d2i, OSSL_HTTP_REQ_CTX_exchange, OSSL_HTTP_REQ_CTX_get0_mem_bio, OSSL_HTTP_REQ_CTX_get_resp_len, OSSL_HTTP_REQ_CTX_set_max_response_length, OSSL_HTTP_is_alive - HTTP client low-level functions

SYNOPSIS

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

```
typedef struct ossl_http_req_ctx_st OSSL_HTTP_REQ_CTX;
```

```
OSSL_HTTP_REQ_CTX *OSSL_HTTP_REQ_CTX_new(BIO *wbio, BIO *rbio, int buf_size);
```

```
void OSSL_HTTP_REQ_CTX_free(OSSL_HTTP_REQ_CTX *rctx);
```

```
int OSSL_HTTP_REQ_CTX_set_request_line(OSSL_HTTP_REQ_CTX *rctx, int method_POST,  
                                       const char *server, const char *port,  
                                       const char *path);
```

```

int OSSL_HTTP_REQ_CTX_add1_header(OSSL_HTTP_REQ_CTX *rctx,
    const char *name, const char *value);

int OSSL_HTTP_REQ_CTX_set_expected(OSSL_HTTP_REQ_CTX *rctx,
    const char *content_type, int asn1,
    int timeout, int keep_alive);

int OSSL_HTTP_REQ_CTX_set1_req(OSSL_HTTP_REQ_CTX *rctx, const char *content_type,
    const ASN1_ITEM *it, const ASN1_VALUE *req);

int OSSL_HTTP_REQ_CTX_nbio(OSSL_HTTP_REQ_CTX *rctx);

int OSSL_HTTP_REQ_CTX_nbio_d2i(OSSL_HTTP_REQ_CTX *rctx,
    ASN1_VALUE **pval, const ASN1_ITEM *it);

BIO *OSSL_HTTP_REQ_CTX_exchange(OSSL_HTTP_REQ_CTX *rctx);

BIO *OSSL_HTTP_REQ_CTX_get0_mem_bio(const OSSL_HTTP_REQ_CTX *rctx);

size_t OSSL_HTTP_REQ_CTX_get_resp_len(const OSSL_HTTP_REQ_CTX *rctx);

void OSSL_HTTP_REQ_CTX_set_max_response_length(OSSL_HTTP_REQ_CTX *rctx,
    unsigned long len);

int OSSL_HTTP_is_alive(const OSSL_HTTP_REQ_CTX *rctx);

```

DESCRIPTION

OSSL_HTTP_REQ_CTX is a context structure for an HTTP request and response, used to collect all the necessary data to perform that request.

This file documents low-level HTTP functions rarely used directly.

High-level HTTP client functions like OSSL_HTTP_get(3) and OSSL_HTTP_transfer(3) should be preferred.

OSSL_HTTP_REQ_CTX_new() allocates a new HTTP request context structure, which gets populated with the BIO to write/send the request to (wbio), the BIO to read/receive the response from (rbio, which may be equal to wbio), and the maximum expected response header line length buf_size.

A value ≤ 0 indicates that the `OSSL_HTTP_DEFAULT_MAX_LINE_LEN` of 4KiB should be used. `buf_size` is also used as the number of content bytes that are read at a time. The allocated context structure includes an internal memory BIO, which collects the HTTP request header lines.

`OSSL_HTTP_REQ_CTX_free()` frees up the HTTP request context `rctx`. The `rbio` is not free'd, `wbio` will be free'd if `free_wbio` is set.

`OSSL_HTTP_REQ_CTX_set_request_line()` adds the HTTP request line to the context. The HTTP method is determined by `method_POST`, which should be 1 to indicate "POST" or 0 to indicate "GET". `server` and `port` may be set to indicate a proxy server and port that the request should go through, otherwise they should be left NULL. `path` is the HTTP request path; if left NULL, "/" is used.

`OSSL_HTTP_REQ_CTX_add1_header()` adds header name with value `value` to the context `rctx`. It can be called more than once to add multiple header lines. For example, to add a "Host" header for "example.com" you would call:

```
OSSL_HTTP_REQ_CTX_add1_header(ctx, "Host", "example.com");
```

`OSSL_HTTP_REQ_CTX_set_expected()` optionally sets in `rctx` some expectations of the HTTP client on the response. Due to the structure of an HTTP request, if the `keep_alive` argument is nonzero the function must be used before calling `OSSL_HTTP_REQ_CTX_set1_req()`. If the `content_type` parameter is not NULL then the client will check that the given content type string is included in the HTTP header of the response and return an error if not. If the `asn1` parameter is nonzero a structure in ASN.1 encoding will be expected as the response content and input streaming is disabled. This means that an ASN.1 sequence header is required, its length field is checked, and

`OSSL_HTTP_REQ_CTX_get0_mem_bio()` should be used to get the buffered

response. Otherwise (by default) any input format is allowed without length checks. In this case the BIO given as rbio argument to `OSSL_HTTP_REQ_CTX_new()` should be used directly to read the response contents, which may support streaming. If the timeout parameter is > 0 this indicates the maximum number of seconds the subsequent HTTP transfer (sending the request and receiving a response) is allowed to take. `timeout == 0` enables waiting indefinitely, i.e., no timeout can occur. This is the default. `timeout < 0` takes over any value set via the `overall_timeout` argument of `OSSL_HTTP_open(3)` with the default being 0, which means no timeout. If the `keep_alive` parameter is 0, which is the default, the connection is not kept open after receiving a response. This is the default behavior for HTTP 1.0. If the value is 1 or 2 then a persistent connection is requested. If the value is 2 then a persistent connection is required, i.e., an error occurs in case the server does not grant it.

`OSSL_HTTP_REQ_CTX_set1_req()` finalizes the HTTP request context. It is needed if the `method_POST` parameter in the `OSSL_HTTP_REQ_CTX_set_request_line()` call was 1 and an ASN.1-encoded request should be sent. It must also be used when requesting "keep-alive", even if a GET request is going to be sent, in which case `req` must be NULL. Unless `req` is NULL, the function adds the DER encoding of `req` using the ASN.1 template it to do the encoding (which does not support streaming). The HTTP header "Content-Length" is filled out with the length of the request. `content_type` must be NULL if `req` is NULL. If `content_type` isn't NULL, the HTTP header "Content-Type" is also added with the given string value. The header lines are added to the internal memory BIO for the request header.

`OSSL_HTTP_REQ_CTX_nbio()` attempts to send the request prepared in `rctx` and to gather the response via HTTP, using the `wbio` and `rbio` that were given when calling `OSSL_HTTP_REQ_CTX_new()`. The function may need to be called again if its result is -1, which indicates

BIO_should_retry(3). In such a case it is advisable to sleep a little in between, using BIO_wait(3) on the read BIO to prevent a busy loop.

OSSL_HTTP_REQ_CTX_nbio_d2i() is like OSSL_HTTP_REQ_CTX_nbio() but on success in addition parses the response, which must be a DER-encoded ASN.1 structure, using the ASN.1 template it and places the result in *pval.

OSSL_HTTP_REQ_CTX_exchange() calls OSSL_HTTP_REQ_CTX_nbio() as often as needed in order to exchange a request and response or until a timeout is reached. On success it returns a pointer to the BIO that can be used to read the result. If an ASN.1-encoded response was expected, this is the BIO returned by OSSL_HTTP_REQ_CTX_get0_mem_bio() when called after the exchange. This memory BIO does not support streaming. Otherwise the returned BIO is the rbio given to OSSL_HTTP_REQ_CTX_new(), which may support streaming. When this BIO is returned, it has been read past the end of the response header, such that the actual response body can be read from it. The returned BIO pointer MUST NOT be freed by the caller.

OSSL_HTTP_REQ_CTX_get0_mem_bio() returns the internal memory BIO. Before the HTTP request is sent, this could be used to adapt its header lines. Use with caution! After receiving a response via HTTP, the BIO represents the current state of reading the response header. If the response was expected to be ASN.1 encoded, its contents can be read via this BIO, which does not support streaming. The returned BIO pointer must not be freed by the caller.

OSSL_HTTP_REQ_CTX_get_resp_len() returns the size of the response contents in rctx if provided by the server as <Content-Length> header field, else 0.

OSSL_HTTP_REQ_CTX_set_max_response_length() sets the maximum allowed

response content length for rctx to len. If not set or len is 0 then the OSSL_HTTP_DEFAULT_MAX_RESP_LEN is used, which currently is 100 KiB. If the "Content-Length" header is present and exceeds this value or the content is an ASN.1 encoded structure with a length exceeding this value or both length indications are present but disagree then an error occurs.

OSSL_HTTP_is_alive() can be used to query if the HTTP connection given by rctx is still alive, i.e., has not been closed. It returns 0 if rctx is NULL.

If the client application requested or required a persistent connection and this was granted by the server, it can keep rctx as long as it wants to send further requests and OSSL_HTTP_is_alive() returns nonzero, else it should call OSSL_HTTP_REQ_CTX_free(rctx) or OSSL_HTTP_close(3). In case the client application keeps rctx but the connection then dies for any reason at the server side, it will notice this obtaining an I/O error when trying to send the next request via rctx.

WARNINGS

The server's response may be unexpected if the hostname that was used to create the wbio, any "Host" header, and the host specified in the request URL do not match.

Many of these functions must be called in a certain order.

First, the HTTP request context must be allocated:

```
OSSL_HTTP_REQ_CTX_new().
```

Then, the HTTP request must be prepared with request data:

1. Calling OSSL_HTTP_REQ_CTX_set_request_line().

2. Adding extra header lines with `OSSL_HTTP_REQ_CTX_add1_header()`.

This is optional and may be done multiple times with different names.

3. Finalize the request using `OSSL_HTTP_REQ_CTX_set1_req()`. This may be omitted if the GET method is used and "keep-alive" is not requested.

When the request context is fully prepared, the HTTP exchange may be performed with `OSSL_HTTP_REQ_CTX_nbio()` or `OSSL_HTTP_REQ_CTX_exchange()`.

RETURN VALUES

`OSSL_HTTP_REQ_CTX_new()` returns a pointer to a `OSSL_HTTP_REQ_CTX`, or NULL on error.

`OSSL_HTTP_REQ_CTX_free()` and

`OSSL_HTTP_REQ_CTX_set_max_response_length()` do not return values.

`OSSL_HTTP_REQ_CTX_set_request_line()`, `OSSL_HTTP_REQ_CTX_add1_header()`, `OSSL_HTTP_REQ_CTX_set1_req()`, and `OSSL_HTTP_REQ_CTX_set_expected()` return 1 for success and 0 for failure.

`OSSL_HTTP_REQ_CTX_nbio()` and `OSSL_HTTP_REQ_CTX_nbio_d2i()` return 1 for success, 0 on error or redirection, -1 if retry is needed.

`OSSL_HTTP_REQ_CTX_exchange()` and `OSSL_HTTP_REQ_CTX_get0_mem_bio()` return a pointer to a BIO on success as described above or NULL on failure. The returned BIO must not be freed by the caller.

`OSSL_HTTP_REQ_CTX_get_resp_len()` returns the size of the response contents or 0 if not available or an error occurred.

OSSL_HTTP_is_alive() returns 1 if its argument is non-NULL and the client requested a persistent connection and the server did not disagree on keeping the connection open, else 0.

SEE ALSO

BIO_should_retry(3), BIO_wait(3), ASN1_item_d2i_bio(3),
ASN1_item_i2d_mem_bio(3), OSSL_HTTP_open(3), OSSL_HTTP_get(3),
OSSL_HTTP_transfer(3), OSSL_HTTP_close(3)

HISTORY

The functions described here were added in OpenSSL 3.0.

COPYRIGHT

Copyright 2015-2022 The OpenSSL Project Authors. All Rights Reserved.

Licensed under the Apache License 2.0 (the "License"). You may not use this file except in compliance with the License. You can obtain a copy in the file LICENSE in the source distribution or at <https://www.openssl.org/source/license.html>.

3.0.7 2023-07-13 OSSL_HTTP_REQ_CTX(3openssl)