

NAME

OSSL_HTTP_open, OSSL_HTTP_bio_cb_t, OSSL_HTTP_proxy_connect,
 OSSL_HTTP_set1_request, OSSL_HTTP_exchange, OSSL_HTTP_get, OSSL_HTTP_transfer,
 OSSL_HTTP_close - HTTP client high-level functions

SYNOPSIS

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

```
typedef BIO *(*OSSL_HTTP_bio_cb_t)(BIO *bio, void *arg,
    int connect, int detail);
OSSL_HTTP_REQ_CTX *OSSL_HTTP_open(const char *server, const char *port,
    const char *proxy, const char *no_proxy,
    int use_ssl, BIO *bio, BIO *rbio,
    OSSL_HTTP_bio_cb_t bio_update_fn, void *arg,
    int buf_size, int overall_timeout);
int OSSL_HTTP_proxy_connect(BIO *bio, const char *server, const char *port,
    const char *proxyuser, const char *proxypass,
    int timeout, BIO *bio_err, const char *prog);
int OSSL_HTTP_set1_request(OSSL_HTTP_REQ_CTX *rctx, const char *path,
    const STACK_OF(CONF_VALUE) *headers,
    const char *content_type, BIO *req,
    const char *expected_content_type, int expect_asn1,
    size_t max_resp_len, int timeout, int keep_alive);
BIO *OSSL_HTTP_exchange(OSSL_HTTP_REQ_CTX *rctx, char **redirection_url);
BIO *OSSL_HTTP_get(const char *url, const char *proxy, const char *no_proxy,
    BIO *bio, BIO *rbio,
    OSSL_HTTP_bio_cb_t bio_update_fn, void *arg,
    int buf_size, const STACK_OF(CONF_VALUE) *headers,
    const char *expected_content_type, int expect_asn1,
    size_t max_resp_len, int timeout);
BIO *OSSL_HTTP_transfer(OSSL_HTTP_REQ_CTX **prctx,
    const char *server, const char *port,
    const char *path, int use_ssl,
    const char *proxy, const char *no_proxy,
    BIO *bio, BIO *rbio,
    OSSL_HTTP_bio_cb_t bio_update_fn, void *arg,
    int buf_size, const STACK_OF(CONF_VALUE) *headers,
    const char *content_type, BIO *req,
    const char *expected_content_type, int expect_asn1,
    size_t max_resp_len, int timeout, int keep_alive);
```

```
int OSSL_HTTP_close(OSSL_HTTP_REQ_CTX *rctx, int ok);
```

DESCRIPTION

OSSL_HTTP_open() initiates an HTTP session using the *bio* argument if not NULL, else by connecting to a given *server* optionally via a *proxy*.

Typically the OpenSSL build supports sockets and the *bio* parameter is NULL. In this case *rbio* must be NULL as well and the *server* must be non-NULL. The function creates a network BIO internally using **BIO_new_connect(3)** for connecting to the given server and the optionally given *port*, defaulting to 80 for HTTP or 443 for HTTPS. Then this internal BIO is used for setting up a connection and for exchanging one or more request and response. If *bio* is given and *rbio* is NULL then this *bio* is used instead. If both *bio* and *rbio* are given (which may be memory BIOs for instance) then no explicit connection is set up, but *bio* is used for writing requests and *rbio* for reading responses. As soon as the client has flushed *bio* the server must be ready to provide a response or indicate a waiting condition via *rbio*.

If *bio* is given, it is an error to provide *proxy* or *no_proxy* arguments, while *server* and *port* arguments may be given to support diagnostic output. If *bio* is NULL the optional *proxy* parameter can be used to set an HTTP(S) proxy to use (unless overridden by "no_proxy" settings). If TLS is not used this defaults to the environment variable "http_proxy" if set, else "HTTP_PROXY". If *use_ssl* != 0 it defaults to "https_proxy" if set, else "HTTPS_PROXY". An empty proxy string "" forbids using a proxy. Else the format is "[http[s]://][userinfo@]host[:port][/path][?query][#fragment]", where any userinfo, path, query, and fragment given is ignored. The default proxy port number is 80, or 443 in case "https:" is given. The HTTP client functions connect via the given proxy unless the *server* is found in the optional list *no_proxy* of proxy hostnames (if not NULL; default is the environment variable "no_proxy" if set, else "NO_PROXY"). Proxying plain HTTP is supported directly, while using a proxy for HTTPS connections requires a suitable callback function such as **OSSL_HTTP_proxy_connect()**, described below.

If *use_ssl* is nonzero a TLS connection is requested and the *bio_update_fn* parameter must be provided.

The parameter *bio_update_fn*, which is optional if *use_ssl* is 0, may be used to modify the connection BIO used by the HTTP client, but cannot be used when both *bio* and *rbio* are given. *bio_update_fn* is a BIO connect/disconnect callback function with prototype

```
BIO *(*OSSL_HTTP_bio_cb_t)(BIO *bio, void *arg, int connect, int detail)
```

The callback function may modify the BIO provided in the *bio* argument, whereby it may make use of a custom defined argument *arg*, which may for instance point to an **SSL_CTX** structure. During connection establishment, just after calling **BIO_do_connect_retry()**, the callback function is invoked

with the *connect* argument being 1 and *detail* being 1 if *use_ssl* is nonzero (i.e., HTTPS is requested), else 0. On disconnect *connect* is 0 and *detail* is 1 if no error occurred, else 0. For instance, on connect the callback may push an SSL BIO to implement HTTPS; after disconnect it may do some diagnostic output and pop and free the SSL BIO.

The callback function must return either the potentially modified BIO *bio*. or NULL to indicate failure, in which case it should not modify the BIO.

Here is a simple example that supports TLS connections (but not via a proxy):

```
BIO *http_tls_cb(BIO *bio, void *arg, int connect, int detail)
{
    if (connect && detail) { /* connecting with TLS */
        SSL_CTX *ctx = (SSL_CTX *)arg;
        BIO *sbio = BIO_new_ssl(ctx, 1);

        bio = sbio != NULL ? BIO_push(sbio, bio) : NULL;
    } else if (!connect) { /* disconnecting */
        BIO *hbio;

        if (!detail) { /* an error has occurred */
            /* optionally add diagnostics here */
        }
        BIO_ssl_shutdown(bio);
        hbio = BIO_pop(bio);
        BIO_free(bio); /* SSL BIO */
        bio = hbio;
    }
    return bio;
}
```

After disconnect the modified BIO will be deallocated using **BIO_free_all()**.

The *buf_size* parameter specifies the response header maximum line length. A value ≤ 0 means that the **OSSL_HTTP_DEFAULT_MAX_LINE_LEN** (4KiB) is used. *buf_size* is also used as the number of content bytes that are read at a time.

If the *overall_timeout* parameter is > 0 this indicates the maximum number of seconds the overall HTTP transfer (i.e., connection setup if needed, sending requests, and receiving responses) is allowed to take until completion. A value ≤ 0 enables waiting indefinitely, i.e., no timeout.

OSSL_HTTP_proxy_connect() may be used by an above BIO connect callback function to set up an SSL/TLS connection via an HTTPS proxy. It promotes the given BIO *bio* representing a connection pre-established with a TLS proxy using the HTTP CONNECT method, optionally using proxy client credentials *proxyuser* and *proxypass*, to connect with TLS protection ultimately to *server* and *port*. If the *port* argument is NULL or the empty string it defaults to "443". If the *timeout* parameter is > 0 this indicates the maximum number of seconds the connection setup is allowed to take. A value <= 0 enables waiting indefinitely, i.e., no timeout. Since this function is typically called by applications such as **openssl-s_client(1)** it uses the *bio_err* and *prog* parameters (unless NULL) to print additional diagnostic information in a user-oriented way.

OSSL_HTTP_set1_request() sets up in *rctx* the request header and content data and expectations on the response using the following parameters. If <*rctx*> indicates using a proxy for HTTP (but not HTTPS), the server host (and optionally port) needs to be placed in the header; thus it must be present in *rctx*. For backward compatibility, the server (and optional port) may also be given in the *path* argument beginning with "http://" (thus giving an absoluteURI). If *path* is NULL it defaults to "". If *req* is NULL the HTTP GET method will be used to send the request else HTTP POST with the contents of *req* and optional *content_type*, where the length of the data in *req* does not need to be determined in advance: the BIO will be read on-the-fly while sending the request, which supports streaming. The optional list *headers* may contain additional custom HTTP header lines. If the parameter *expected_content_type* 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 *expect_asn1* parameter is nonzero, a structure in ASN.1 encoding will be expected as response content. The *max_resp_len* parameter specifies the maximum allowed response content length, where the value 0 indicates no limit. 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. A value of 0 enables waiting indefinitely, i.e., no timeout. A value < 0 indicates that the *overall_timeout* parameter value given when opening the HTTP transfer will be used instead. If *keep_alive* is 0 the connection is not kept open after receiving a response, which 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_exchange() exchanges any form of HTTP request and response as specified by *rctx*, which must include both connection and request data, typically set up using **OSSL_HTTP_open()** and **OSSL_HTTP_set1_request()**. It implements the core of the functions described below. If the HTTP method is GET and *redirection_url* is not NULL the latter pointer is used to provide any new location that the server may return with HTTP code 301 (MOVED_PERMANENTLY) or 302 (FOUND). In this case the function returns NULL and the caller is responsible for deallocating the URL with **OPENSSL_free(3)**. If the response header contains one or more "Content-Length" header lines and/or an ASN.1-encoded response is expected, which should include a total length, the length indications received are checked for consistency and for not exceeding any given maximum response length. If an

ASN.1-encoded response is expected, the function returns on success the contents buffered in a memory BIO, which does not support streaming. Otherwise it returns directly the read BIO that holds the response contents, which allows a response of indefinite length and may support streaming. The caller is responsible for freeing the BIO pointer obtained.

OSSL_HTTP_get() uses HTTP GET to obtain data from *bio* if non-NULL, else from the server contained in the *url*, and returns it as a BIO. It supports redirection via HTTP status code 301 or 302. It is meant for transfers with a single round trip, so does not support persistent connections. If *bio* is non-NULL, any host and port components in the *url* are not used for connecting but the hostname is used, as usual, for the "Host" header. Any userinfo and fragment components in the *url* are ignored. Any query component is handled as part of the path component. If the scheme component of the *url* is "https" a TLS connection is requested and the *bio_update_fn*, as described for **OSSL_HTTP_open()**, must be provided. Also the remaining parameters are interpreted as described for **OSSL_HTTP_open()** and **OSSL_HTTP_set1_request()**, respectively. The caller is responsible for freeing the BIO pointer obtained.

OSSL_HTTP_transfer() exchanges an HTTP request and response over a connection managed via *prctx* without supporting redirection. It combines **OSSL_HTTP_open()**, **OSSL_HTTP_set1_request()**, **OSSL_HTTP_exchange()**, and **OSSL_HTTP_close()**. If *prctx* is not NULL it reuses any open connection represented by a non-NULL **prctx*. It keeps the connection open if a persistent connection is requested or required and this was granted by the server, else it closes the connection and assigns NULL to **prctx*. The remaining parameters are interpreted as described for **OSSL_HTTP_open()** and **OSSL_HTTP_set1_request()**, respectively. The caller is responsible for freeing the BIO pointer obtained.

OSSL_HTTP_close() closes the connection and releases *rctx*. The *ok* parameter is passed to any BIO update function given during setup as described above for **OSSL_HTTP_open()**. It must be 1 if no error occurred during the HTTP transfer and 0 otherwise.

NOTES

The names of the environment variables used by this implementation: "http_proxy", "HTTP_PROXY", "https_proxy", "HTTPS_PROXY", "no_proxy", and "NO_PROXY", have been chosen for maximal compatibility with other HTTP client implementations such as wget, curl, and git.

RETURN VALUES

OSSL_HTTP_open() returns on success a **OSSL_HTTP_REQ_CTX**, else NULL.

OSSL_HTTP_proxy_connect() and **OSSL_HTTP_set1_request()** return 1 on success, 0 on error.

On success, **OSSL_HTTP_exchange()**, **OSSL_HTTP_get()**, and **OSSL_HTTP_transfer()** return a

memory BIO that buffers all the data received if an ASN.1-encoded response is expected, otherwise a BIO that may support streaming. The BIO must be freed by the caller. On failure, they return NULL. Failure conditions include connection/transfer timeout, parse errors, etc. The caller is responsible for freeing the BIO pointer obtained.

OSSL_HTTP_close() returns 0 if anything went wrong while disconnecting, else 1.

SEE ALSO

OSSL_HTTP_parse_url(3), **BIO_new_connect(3)**, **ASN1_item_i2d_mem_bio(3)**, **ASN1_item_d2i_bio(3)**, **OSSL_HTTP_is_alive(3)**

HISTORY

All the functions described here were added in OpenSSL 3.0.

COPYRIGHT

Copyright 2019-2023 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>.