NAME

CURLOPT_SSL_CTX_FUNCTION − SSL context callback

SYNOPSIS

```
#include <curl/curl.h>

CURLcode ssl_ctx_callback(CURL *curl, void *ssl_ctx, void *clientp);

CURLcode curl_easy_setopt(CURL *handle, CURLOPT_SSL_CTX_FUNCTION,
                          ssl_ctx_callback);
```

DESCRIPTION

Pass a pointer to your callback function, which should match the prototype shown above.

This callback function gets called by libcurl immediately before the initialization of an SSL connection after having processed all other SSL related options to give a last chance to an application to modify the behavior of the SSL initialization. The *ssl_ctx* parameter is a pointer to the SSL library's *SSL_CTX* for OpenSSL or wolfSSL, a pointer to *mbedtls_ssl_config* for mbedTLS. If an error is returned from the callback no attempt to establish a connection is made and the perform operation returns the callback's error code. Set the *clientp* argument passed in to this callback with the *CURLOPT_SSL_CTX_DATA(3)* option.

This function gets called for all new connections made to a server, during the SSL negotiation. While *ssl_ctx* points to a newly initialized object each time, the pointer may still be the same as in a prior call.

To use this callback, a non−trivial amount of knowledge of your SSL library is necessary. For example, you can use this function to call library−specific callbacks to add additional validation code for certificates, and even to change the actual URI of an HTTPS request.

For OpenSSL, asynchronous certificate verification via *SSL_set_retry_verify* is supported. When *SSL_set_retry_verify* is set, the transfer is paused. When verification should continue, call *curl_easy_pause(3)* to unpause the transfer. (Added in 8.3.0, Pausing added in 8.16.0)

The *CURLOPT_SSL_CTX_FUNCTION(3)* callback allows the application to reach in and modify SSL details in the connection without libcurl itself knowing anything about it, which then subsequently can lead to libcurl unknowingly reusing SSL connections with different properties. To remedy this you may set *CURLOPT_FORBID_REUSE(3)* from the callback function.

A connection that is set up with this callback can be put in the connection pool by libcurl and then reused in following transfers without the callback being called. The connection may even be selected from the pool to be used for transfers not using this callback. If the callback should only be valid for the specific transfer the callback verifies, it should be marked unsuitable for reuse with *CURLOPT_FORBID_REUSE(3)*.

If you are using DNS−over−HTTPS (DoH) via *CURLOPT_DOH_URL(3)* then this callback is also called for those transfers and the curl handle is set to an internal handle. **This behavior is subject to change.** We recommend setting *CURLOPT_PRIVATE(3)* on your curl handle so you can identify it correctly in the context callback. If you have a reason to modify DoH SSL context please let us know on the curl−library mailing list because we are considering removing this capability.

libcurl does not guarantee the lifetime of the passed in object once this callback function has returned. Your application must not assume that it can keep using the SSL context or data derived from it once this function is completed.

For libcurl builds using TLS backends that support CA caching and *CURLOPT_CA_CACHE_TIMEOUT(3)* is not set to zero, multiple calls to this callback may be done with the same CA store in memory.

DEFAULT

NULL

PROTOCOLS

This functionality affects all TLS based protocols: HTTPS, FTPS, IMAPS, POP3S, SMTPS etc.

This option works only with the following TLS backends: OpenSSL, mbedTLS and wolfSSL

EXAMPLE

```
/* OpenSSL specific */

#include <openssl/ssl.h>
#include <curl/curl.h>
#include <stdio.h>

static CURLcode sslctx_function(CURL *curl, void *sslctx, void *pointer)
{
  X509_STORE *store;
  X509 *cert = NULL;
  BIO *bio;
  char *mypem = pointer;
  /* get a BIO */
  bio = BIO_new_mem_buf(mypem, -1);
  /* use it to read the PEM formatted certificate from memory into an
   * X509 structure that SSL can use
   */
  PEM_read_bio_X509(bio, &cert, 0, NULL);
  if(!cert)
    printf("PEM_read_bio_X509 failed...\n");

  /* get a pointer to the X509 certificate store (which may be empty) */
  store = SSL_CTX_get_cert_store((SSL_CTX *)sslctx);

  /* add our certificate to this store */
  if(X509_STORE_add_cert(store, cert) == 0)
    printf("error adding certificate\n");

  /* decrease reference counts */
  X509_free(cert);
  BIO_free(bio);

  /* all set to go */
  return CURLE_OK;
}

int main(void)
{
  CURL *curl;
  CURLcode result;
  /* CA cert in PEM format, replace the XXXs */
  char *mypem =
    "-----BEGIN CERTIFICATE-----\n"
    "XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX\n"
    "XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX\n"
    "XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX\n"
    "XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX\n"
```

```
  "XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX\n"
  "XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX\n"
  "-----END CERTIFICATE-----\n";

 curl_global_init(CURL_GLOBAL_ALL);
 curl = curl_easy_init();

 curl_easy_setopt(curl, CURLOPT_SSLCERTTYPE, "PEM");
 curl_easy_setopt(curl, CURLOPT_SSL_VERIFYPEER, 1L);
 curl_easy_setopt(curl, CURLOPT_URL, "https://www.example.com/");

 curl_easy_setopt(curl, CURLOPT_SSL_CTX_FUNCTION, *sslctx_function);
 curl_easy_setopt(curl, CURLOPT_SSL_CTX_DATA, mypem);
 result = curl_easy_perform(curl);
 if(result == CURLE_OK)
   printf("*** transfer succeeded ***\n");
 else
   printf("*** transfer failed ***\n");

 curl_easy_cleanup(curl);
 curl_global_cleanup();
 return (int)result;
}
```

AVAILABILITY

Added in curl 7.10.6

RETURN VALUE

CURLE_OK if supported; or an error such as:

CURLE_NOT_BUILT_IN − Not supported by the SSL backend

SEE ALSO

CURLOPT_CAINFO(3), **CURLOPT_CAINFO_BLOB**(3), **CURLOPT_CA_CACHE_TIMEOUT**(3), **CURLOPT_SSL_CTX_DATA**(3), **CURLOPT_SSL_VERIFYHOST**(3), **CURLOPT_SSL_VERIFYPEER**(3)