NAME

curl_easy_setopt − set options for a curl easy handle

SYNOPSIS

#include <curl/curl.h>

CURLcode curl_easy_setopt(CURL *handle, CURLoption option, parameter);

DESCRIPTION

curl_easy_setopt() is used to tell libcurl how to behave. By using the appropriate options to *curl_easy_setopt*, you can change libcurl's behavior. All options are set with the *option* followed by a *parameter*. That parameter can be a **long**, a **function pointer**, an **object pointer** or a **curl_off_t**, depending on what the specific option expects. Read this manual carefully as bad input values may cause libcurl to behave badly! You can only set one option in each function call. A typical application uses many curl_easy_setopt() calls in the setup phase.

Options set with this function call are valid for all forthcoming transfers performed using this *handle*. The options are not in any way reset between transfers, so if you want subsequent transfers with different options, you must change them between the transfers. You can optionally reset all options back to internal default with *curl_easy_reset(3)*.

Strings passed to libcurl as 'char *' arguments, will not be copied by the library. Instead you should keep them available until libcurl no longer needs them. Failing to do so will cause very odd behavior or even crashes. libcurl will need them until you call *curl_easy_cleanup(3)* or you set the same option again to use a different pointer.

The *handle* is the return code from a *curl_easy_init(3)* or *curl_easy_duphandle(3)* call.

BEHAVIOR OPTIONS

CURLOPT_VERBOSE

Set the parameter to non-zero to get the library to display a lot of verbose information about its operations. Very useful for libcurl and/or protocol debugging and understanding. The verbose information will be sent to stderr, or the stream set with *CURLOPT_STDERR*.

You hardly ever want this set in production use, you will almost always want this when you debug/report problems. Another neat option for debugging is the *CURLOPT_DEBUGFUNCTION*.

CURLOPT_HEADER

A non-zero parameter tells the library to include the header in the body output. This is only relevant for protocols that actually have headers preceding the data (like HTTP).

CURLOPT_NOPROGRESS

A non-zero parameter tells the library to shut off the built-in progress meter completely.

Future versions of libcurl is likely to not have any built-in progress meter at all.

CURLOPT_NOSIGNAL

Pass a long. If it is non-zero, libcurl will not use any functions that install signal handlers or any functions that cause signals to be sent to the process. This option is mainly here to allow multi-threaded unix applications to still set/use all timeout options etc, without risking getting signals. (Added in 7.10)

Consider building libcurl with ares support to enable asynchronous DNS lookups. It enables nice timeouts for name resolves without signals.

CALLBACK OPTIONS

CURLOPT_WRITEFUNCTION

Function pointer that should match the following prototype: **size_t function(void *ptr, size_t size, size_t nmemb, void *stream);** This function gets called by libcurl as soon as there is data received that needs to be saved. The size of the data pointed to by *ptr* is *size* multiplied with *nmemb*, it will not be zero terminated. Return the number of bytes actually taken care of. If that amount differs from the amount passed to your function, it'll signal an error to the library and it will abort the transfer and return *CURLE_WRITE_ERROR*.

This function may be called with zero bytes data if the transfered file is empty.

Set this option to NULL to get the internal default function. The internal default function will write the data to the FILE * given with *CURLOPT_WRITEDATA*.

Set the *stream* argument with the *CURLOPT_WRITEDATA* option.

The callback function will be passed as much data as possible in all invokes, but you cannot possibly make any assumptions. It may be one byte, it may be thousands. The maximum amount of data that can be passed to the write callback is defined in the curl.h header file: CURL_MAX_WRITE_SIZE.

CURLOPT_WRITEDATA

Data pointer to pass to the file write function. If you use the *CURLOPT_WRITEFUNCTION* option, this is the pointer you'll get as input. If you don't use a callback, you must pass a 'FILE *' as libcurl will pass this to fwrite() when writing data.

The internal *CURLOPT_WRITEFUNCTION* will write the data to the FILE * given with this option, or to stdout if this option hasn't been set.

If you're using libcurl as a win32 DLL, you **MUST** use the *CURLOPT_WRITEFUNCTION* if you set this option or you will experience crashes.

This option is also known with the older name *CURLOPT_FILE*, the name *CURLOPT_WRITEDATA* was introduced in 7.9.7.

CURLOPT_READFUNCTION

Function pointer that should match the following prototype: **size_t function(void *ptr, size_t size, size_t nmemb, void *stream);** This function gets called by libcurl as soon as it needs to read data in order to send it to the peer. The data area pointed at by the pointer *ptr* may be filled with at most *size* multiplied with *nmemb* number of bytes. Your function must return the actual number of bytes that you stored in that memory area. Returning 0 will signal end-of-file to the library and cause it to stop the current transfer.

If you stop the current transfer by returning 0 "pre-maturely" (i.e before the server expected it, like when you've told you will upload N bytes and you upload less than N bytes), you may experience that the server "hangs" waiting for the rest of the data that won't come.

The read callback may return *CURL_READFUNC_ABORT* to stop the current operation immediately, resulting in a *CURLE_ABORTED_BY_CALLBACK* error code from the transfer (Added in 7.12.1)

If you set the callback pointer to NULL, or doesn't set it at all, the default internal read function will be used. It is simply doing an fread() on the FILE * stream set with *CURLOPT_READDATA*.

CURLOPT_READDATA

Data pointer to pass to the file read function. If you use the *CURLOPT_READFUNCTION* option, this is the pointer you'll get as input. If you don't specify a read callback but instead rely on the

default internal read function, this data must be a valid readable FILE *.

If you're using libcurl as a win32 DLL, you MUST use a *CURLOPT_READFUNCTION* if you set this option.

This option is also known with the older name *CURLOPT_INFILE*, the name *CURLOPT_READDATA* was introduced in 7.9.7.

CURLOPT_IOCTLFUNCTION

Function pointer that should match the *curl_ioctl_callback* prototype found in <*curl/curl.h*>. This function gets called by libcurl when something special I/O-related needs to be done that the library can't do by itself. For now, rewinding the read data stream is the only action it can request. The rewinding of the read data stream may be necessary when doing a HTTP PUT or POST with a multi-pass authentication method. (Opion added in 7.12.3)

CURLOPT_IOCTLDATA

Pass a pointer that will be untouched by libcurl and passed as the 3rd argument in the ioctl callback set with *CURLOPT_IOCTLFUNCTION*. (Option added in 7.12.3)

CURLOPT_PROGRESSFUNCTION

Function pointer that should match the *curl_progress_callback* prototype found in <*curl/curl.h*>. This function gets called by libcurl instead of its internal equivalent with a frequent interval during data transfer (roughly once per second). Unknown/unused argument values pass to the callback will be set to zero (like if you only download data, the upload size will remain 0). Returning a non-zero value from this callback will cause libcurl to abort the transfer and return *CURLE_ABORTED_BY_CALLBACK*.

If you transfer data with the multi interface, this function will not be called during periods of idleness unless you call the appropriate libcurl function that performs transfers. Usage of the **CURLOPT_PROGRESSFUNCTION** callback is not recommended when using the multi interface.

CURLOPT_NOPROGRESS must be set to FALSE to make this function actually get called.

CURLOPT_PROGRESSDATA

Pass a pointer that will be untouched by libcurl and passed as the first argument in the progress callback set with *CURLOPT_PROGRESSFUNCTION*.

CURLOPT_HEADERFUNCTION

Function pointer that should match the following prototype: *size_t function(void *ptr, size_t size, size_t nmemb, void *stream);*. This function gets called by libcurl as soon as it has received header data. The header callback will be called once for each header and only complete header lines are passed on to the callback. Parsing headers should be easy enough using this. The size of the data pointed to by *ptr* is *size* multiplied with *nmemb*. Do not assume that the header line is zero terminated! The pointer named *stream* is the one you set with the *CURLOPT_WRITEHEADER* option. The callback function must return the number of bytes actually taken care of, or return -1 to signal error to the library (it will cause it to abort the transfer with a *CURLE_WRITE_ERROR* return code).

Since 7.14.1: When a server sends a chunked encoded transfer, it may contain a trailer. That trailer is identical to a HTTP header and if such a trailer is received it is passed to the application using this callback as well. There are several ways to detect it being a trailer and not an ordinary header: 1) it comes after the response-body. 2) it comes after the final header line (CR LF) 3) a Trailer: header among the response-headers mention what header to expect in the trailer.

CURLOPT_WRITEHEADER

(This option is also known as **CURLOPT_HEADERDATA**) Pass a pointer to be used to write the header part of the received data to. If you don't use your own callback to take care of the writing, this must be a valid FILE *. See also the *CURLOPT_HEADERFUNCTION* option above on how

to set a custom get-all-headers callback.

CURLOPT_DEBUGFUNCTION
: Function pointer that should match the following prototype: *int curl_debug_callback (CURL *, curl_infotype, char *, size_t, void *); CURLOPT_DEBUGFUNCTION* replaces the standard debug function used when *CURLOPT_VERBOSE* is in effect. This callback receives debug information, as specified with the **curl_infotype** argument. This function must return 0. The data pointed to by the char * passed to this function WILL NOT be zero terminated, but will be exactly of the size as told by the size_t argument.

  Available curl_infotype values:

  CURLINFO_TEXT
  : The data is informational text.

  CURLINFO_HEADER_IN
  : The data is header (or header-like) data received from the peer.

  CURLINFO_HEADER_OUT
  : The data is header (or header-like) data sent to the peer.

  CURLINFO_DATA_IN
  : The data is protocol data received from the peer.

  CURLINFO_DATA_OUT
  : The data is protocol data sent to the peer.

CURLOPT_DEBUGDATA
: Pass a pointer to whatever you want passed in to your *CURLOPT_DEBUGFUNCTION* in the last void * argument. This pointer is not used by libcurl, it is only passed to the callback.

CURLOPT_SSL_CTX_FUNCTION
: Function pointer that should match the following prototype: **CURLcode sslctxfun(CURL *curl, void *sslctx, void *parm);** This function gets called by libcurl just 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 behaviour of openssl's ssl initialization. The *sslctx* parameter is actually a pointer to an openssl *SSL_CTX*. If an error is returned no attempt to establish a connection is made and the perform operation will return the error code from this callback function. Set the *parm* argument with the *CURLOPT_SSL_CTX_DATA* option. This option was introduced in 7.11.0.

  This function will get called on all new connections made to a server, during the SSL negotiation. The SSL_CTX pointer will be a new one every time.

  To use this properly, a non-trivial amount of knowledge of the openssl libraries is necessary. Using this function allows for example to use openssl callbacks to add additional validation code for certificates, and even to change the actual URI of an HTTPS request (example used in the lib509 test case). See also the example section for a replacement of the key, certificate and trust file settings.

CURLOPT_SSL_CTX_DATA
: Data pointer to pass to the ssl context callback set by the option *CURLOPT_SSL_CTX_FUNCTION*, this is the pointer you'll get as third parameter, otherwise **NULL**. (Added in 7.11.0)

CURLOPT_CONV_TO_NETWORK_FUNCTION

CURLOPT_CONV_FROM_NETWORK_FUNCTION

CURLOPT_CONV_FROM_UTF8_FUNCTION
: Function pointers that should match the following prototype: CURLcode function(char *ptr, size_t length);

  These three options apply to non-ASCII platforms only. They are available only if

CURL_DOES_CONVERSIONS was defined when libcurl was built. When this is the case, *curl_version_info(3)* will return the CURL_VERSION_CONV feature bit set.

The data to be converted is in a buffer pointed to by the ptr parameter. The amount of data to convert is indicated by the length parameter. The converted data overlays the input data in the buffer pointed to by the ptr parameter. CURLE_OK should be returned upon successful conversion. A CURLcode return value defined by curl.h, such as CURLE_CONV_FAILED, should be returned if an error was encountered.

CURLOPT_CONV_TO_NETWORK_FUNCTION and **CURLOPT_CONV_FROM_NETWORK_FUNCTION** convert between the host encoding and the network encoding. They are used when commands or ASCII data are sent/received over the network.

CURLOPT_CONV_FROM_UTF8_FUNCTION is called to convert from UTF8 into the host encoding. It is required only for SSL processing.

If you set a callback pointer to NULL, or don't set it at all, the built-in libcurl iconv functions will be used. If HAVE_ICONV was not defined when libcurl was built, and no callback has been established, conversion will return the CURLE_CONV_REQD error code.

If HAVE_ICONV is defined, CURL_ICONV_CODESET_OF_HOST must also be defined. For example:

```
#define CURL_ICONV_CODESET_OF_HOST "IBM-1047"
```

The iconv code in libcurl will default the network and UTF8 codeset names as follows:

```
#define CURL_ICONV_CODESET_OF_NETWORK "ISO8859-1"

#define CURL_ICONV_CODESET_FOR_UTF8   "UTF-8"
```

You will need to override these definitions if they are different on your system.

ERROR OPTIONS

CURLOPT_ERRORBUFFER

Pass a char * to a buffer that the libcurl may store human readable error messages in. This may be more helpful than just the return code from *curl_easy_perform*. The buffer must be at least CURL_ERROR_SIZE big.

Use *CURLOPT_VERBOSE* and *CURLOPT_DEBUGFUNCTION* to better debug/trace why errors happen.

If the library does not return an error, the buffer may not have been touched. Do not rely on the contents in those cases.

CURLOPT_STDERR

Pass a FILE * as parameter. Tell libcurl to use this stream instead of stderr when showing the progress meter and displaying *CURLOPT_VERBOSE* data.

CURLOPT_FAILONERROR

A non-zero parameter tells the library to fail silently if the HTTP code returned is equal to or larger than 400. The default action would be to return the page normally, ignoring that code.

NETWORK OPTIONS

CURLOPT_URL

The actual URL to deal with. The parameter should be a char * to a zero terminated string. The string must remain present until curl no longer needs it, as it doesn't copy the string.

If the given URL lacks the protocol part ("http://" or "ftp://" etc), it will attempt to guess which protocol to use based on the given host name. If the given protocol of the set URL is not supported, libcurl will return on error (*CURLE_UNSUPPORTED_PROTOCOL*) when you call *curl_easy_perform(3)* or *curl_multi_perform(3)*. Use *curl_version_info(3)* for detailed info on which protocols that are supported.

The string given to CURLOPT_URL must be url-encoded and following the RFC 2396 (http://curl.haxx.se/rfc/rfc2396.txt).

CURLOPT_URL is the only option that **must** be set before *curl_easy_perform(3)* is called.

CURLOPT_PROXY

Set HTTP proxy to use. The parameter should be a char * to a zero terminated string holding the host name or dotted IP address. To specify port number in this string, append :[port] to the end of the host name. The proxy string may be prefixed with [protocol]:// since any such prefix will be ignored. The proxy's port number may optionally be specified with the separate option *CURLOPT_PROXYPORT*.

When you tell the library to use an HTTP proxy, libcurl will transparently convert operations to HTTP even if you specify an FTP URL etc. This may have an impact on what other features of the library you can use, such as *CURLOPT_QUOTE* and similar FTP specifics that don't work unless you tunnel through the HTTP proxy. Such tunneling is activated with *CURLOPT_HTTPPROXYTUNNEL*.

libcurl respects the environment variables **http_proxy**, **ftp_proxy**, **all_proxy** etc, if any of those is set. The *CURLOPT_PROXY* option does however override any possibly set environment variables.

Starting with 7.14.1, the proxy host string can be specified the exact same way as the proxy environment variables, include protocol prefix (http://) and embedded user + password.

CURLOPT_PROXYPORT

Pass a long with this option to set the proxy port to connect to unless it is specified in the proxy string *CURLOPT_PROXY*.

CURLOPT_PROXYTYPE

Pass a long with this option to set type of the proxy. Available options for this are *CURLPROXY_HTTP*, *CURLPROXY_SOCKS4* (added in 7.15.2) *CURLPROXY_SOCKS5*. The HTTP type is default. (Added in 7.10)

CURLOPT_HTTPPROXYTUNNEL

Set the parameter to non-zero to get the library to tunnel all operations through a given HTTP proxy. There is a big difference between using a proxy and to tunnel through it. If you don't know what this means, you probably don't want this tunneling option.

CURLOPT_INTERFACE

Pass a char * as parameter. This set the interface name to use as outgoing network interface. The name can be an interface name, an IP address or a host name.

CURLOPT_LOCALPORT

Pass a long. This sets the local port number of the socket used for connection. This can be used in combination with *CURLOPT_INTERFACE* and you are recommended to use *CURLOPT_LOCALPORTRANGE* as well when this is set. Note that port numbers are only valid 1 - 65535. (Added in 7.15.2)

CURLOPT_LOCALPORTRANGE

Pass a long. This is the number of attempts libcurl should do to find a working local port number. It starts with the given *CURLOPT_LOCALPORT* and adds one to the number for each retry. Setting this value to 1 or below will make libcurl do only one try for exact port number. Note that port numbers by nature is a scarce resource that will be busy at times so setting this value to something too low might cause unnecessary connection setup failures. (Added in 7.15.2)

CURLOPT_DNS_CACHE_TIMEOUT

Pass a long, this sets the timeout in seconds. Name resolves will be kept in memory for this number of seconds. Set to zero (0) to completely disable caching, or set to -1 to make the cached entries remain forever. By default, libcurl caches this info for 60 seconds.

CURLOPT_DNS_USE_GLOBAL_CACHE

Pass a long. If the value is non-zero, it tells curl to use a global DNS cache that will survive between easy handle creations and deletions. This is not thread-safe and this will use a global variable.

WARNING: this option is considered obsolete. Stop using it. Switch over to using the share interface instead! See *CURLOPT_SHARE* and *curl_share_init(3)*.

CURLOPT_BUFFERSIZE

Pass a long specifying your preferred size (in bytes) for the receive buffer in libcurl. The main point of this would be that the write callback gets called more often and with smaller chunks. This is just treated as a request, not an order. You cannot be guaranteed to actually get the given size. (Added in 7.10)

This size is by default set as big as possible (CURL_MAX_WRITE_SIZE), so it only makse sense to use this option if you want it smaller.

CURLOPT_PORT

Pass a long specifying what remote port number to connect to, instead of the one specified in the URL or the default port for the used protocol.

CURLOPT_TCP_NODELAY

Pass a long specifying whether the TCP_NODELAY option should be set or cleared (1 = set, 0 = clear). The option is cleared by default. This will have no effect after the connection has been established.

Setting this option will disable TCP's Nagle algorithm. The purpose of this algorithm is to try to minimize the number of small packets on the network (where "small packets" means TCP segments less than the Maximum Segment Size (MSS) for the network).

Maximizing the amount of data sent per TCP segment is good because it amortizes the overhead of the send. However, in some cases (most notably telnet or rlogin) small segments may need to be sent without delay. This is less efficient than sending larger amounts of data at a time, and can contribute to congestion on the network if overdone.

NAMES and PASSWORDS OPTIONS (Authentication)

CURLOPT_NETRC

This parameter controls the preference of libcurl between using user names and passwords from your *~/.netrc* file, relative to user names and passwords in the URL supplied with *CURLOPT_URL*.

libcurl uses a user name (and supplied or prompted password) supplied with *CURLOPT_USERPWD* in preference to any of the options controlled by this parameter.

Pass a long, set to one of the values described below.

CURL_NETRC_OPTIONAL
: The use of your ~/.*netrc* file is optional, and information in the URL is to be preferred. The file will be scanned with the host and user name (to find the password only) or with the host only, to find the first user name and password after that *machine*, which ever information is not specified in the URL.

  Undefined values of the option will have this effect.

CURL_NETRC_IGNORED
: The library will ignore the file and use only the information in the URL.

  This is the default.

CURL_NETRC_REQUIRED
: This value tells the library that use of the file is required, to ignore the information in the URL, and to search the file with the host only.

Only machine name, user name and password are taken into account (init macros and similar things aren't supported).

libcurl does not verify that the file has the correct properties set (as the standard Unix ftp client does). It should only be readable by user.

CURLOPT_NETRC_FILE
: Pass a char * as parameter, pointing to a zero terminated string containing the full path name to the file you want libcurl to use as .netrc file. If this option is omitted, and *CURLOPT_NETRC* is set, libcurl will attempt to find the a .netrc file in the current user's home directory. (Added in 7.10.9)

CURLOPT_USERPWD
: Pass a char * as parameter, which should be [user name]:[password] to use for the connection. Use *CURLOPT_HTTPAUTH* to decide authentication method.

  When using NTLM, you can set domain by prepending it to the user name and separating the domain and name with a forward (/) or backward slash (\). Like this: "domain/user:password" or "domain\user:password". Some HTTP servers (on Windows) support this style even for Basic authentication.

  When using HTTP and *CURLOPT_FOLLOWLOCATION*, libcurl might perform several requests to possibly different hosts. libcurl will only send this user and password information to hosts using the initial host name (unless *CURLOPT_UNRESTRICTED_AUTH* is set), so if libcurl follows locations to other hosts it will not send the user and password to those. This is enforced to prevent accidental information leakage.

CURLOPT_PROXYUSERPWD
: Pass a char * as parameter, which should be [user name]:[password] to use for the connection to the HTTP proxy. Use *CURLOPT_PROXYAUTH* to decide authentication method.

CURLOPT_HTTPAUTH
: Pass a long as parameter, which is set to a bitmask, to tell libcurl what authentication method(s) you want it to use. The available bits are listed below. If more than one bit is set, libcurl will first query the site to see what authentication methods it supports and then pick the best one you allow it to use. For some methods, this will induce an extra network round-trip. Set the actual name and password with the *CURLOPT_USERPWD* option. (Added in 7.10.6)

  CURLAUTH_BASIC
  : HTTP Basic authentication. This is the default choice, and the only method that is in wide-spread use and supported virtually everywhere. This is sending the user name and password over the network in plain text, easily captured by others.

CURLAUTH_DIGEST
HTTP Digest authentication. Digest authentication is defined in RFC2617 and is a more secure way to do authentication over public networks than the regular old-fashioned Basic method.

CURLAUTH_GSSNEGOTIATE
HTTP GSS-Negotiate authentication. The GSS-Negotiate (also known as plain "Negotiate") method was designed by Microsoft and is used in their web applications. It is primarily meant as a support for Kerberos5 authentication but may be also used along with another authentication methods. For more information see IETF draft draft-brezak-spnego-http-04.txt.

You need to build libcurl with a suitable GSS-API library for this to work.

CURLAUTH_NTLM
HTTP NTLM authentication. A proprietary protocol invented and used by Microsoft. It uses a challenge-response and hash concept similar to Digest, to prevent the password from being eavesdropped.

You need to build libcurl with OpenSSL support for this option to work, or build libcurl on Windows.

CURLAUTH_ANY
This is a convenience macro that sets all bits and thus makes libcurl pick any it finds suitable. libcurl will automatically select the one it finds most secure.

CURLAUTH_ANYSAFE
This is a convenience macro that sets all bits except Basic and thus makes libcurl pick any it finds suitable. libcurl will automatically select the one it finds most secure.

CURLOPT_PROXYAUTH
Pass a long as parameter, which is set to a bitmask, to tell libcurl what authentication method(s) you want it to use for your proxy authentication. If more than one bit is set, libcurl will first query the site to see what authentication methods it supports and then pick the best one you allow it to use. For some methods, this will induce an extra network round-trip. Set the actual name and password with the *CURLOPT_PROXYUSERPWD* option. The bitmask can be constructed by or'ing together the bits listed above for the *CURLOPT_HTTPAUTH* option. As of this writing, only Basic, Digest and NTLM work. (Added in 7.10.7)

HTTP OPTIONS

CURLOPT_AUTOREFERER
Pass a non-zero parameter to enable this. When enabled, libcurl will automatically set the Referer: field in requests where it follows a Location: redirect.

CURLOPT_ENCODING
Sets the contents of the Accept-Encoding: header sent in an HTTP request, and enables decoding of a response when a Content-Encoding: header is received. Three encodings are supported: *identity*, which does nothing, *deflate* which requests the server to compress its response using the zlib algorithm, and *gzip* which requests the gzip algorithm. If a zero-length string is set, then an Accept-Encoding: header containing all supported encodings is sent.

This is a request, not an order; the server may or may not do it. This option must be set (to any non-NULL value) or else any unsolicited encoding done by the server is ignored. See the special file lib/README.encoding for details.

CURLOPT_FOLLOWLOCATION
A non-zero parameter tells the library to follow any Location: header that the server sends as part of an HTTP header.

This means that the library will re-send the same request on the new location and follow new Location: headers all the way until no more such headers are returned. *CURLOPT_MAXREDIRS* can be used to limit the number of redirects libcurl will follow.

CURLOPT_UNRESTRICTED_AUTH

A non-zero parameter tells the library it can continue to send authentication (user+password) when following locations, even when hostname changed. This option is meaningful only when setting *CURLOPT_FOLLOWLOCATION*.

CURLOPT_MAXREDIRS

Pass a long. The set number will be the redirection limit. If that many redirections have been followed, the next redirect will cause an error (*CURLE_TOO_MANY_REDIRECTS*). This option only makes sense if the *CURLOPT_FOLLOWLOCATION* is used at the same time. Added in 7.15.1: Setting the limit to 0 will make libcurl refuse any redirect. Set it to -1 for an infinite number of redirects (which is the default)

CURLOPT_PUT

A non-zero parameter tells the library to use HTTP PUT to transfer data. The data should be set with *CURLOPT_READDATA* and *CURLOPT_INFILESIZE*.

This option is deprecated and starting with version 7.12.1 you should instead use *CURLOPT_UPLOAD*.

CURLOPT_POST

A non-zero parameter tells the library to do a regular HTTP post. This will also make the library use the a "Content-Type: application/x-www-form-urlencoded" header. (This is by far the most commonly used POST method).

Use the *CURLOPT_POSTFIELDS* option to specify what data to post and *CURLOPT_POSTFIELDSIZE* to set the data size.

Optionally, you can provide data to POST using the *CURLOPT_READFUNCTION* and *CURLOPT_READDATA* options but then you must make sure to not set *CURLOPT_POSTFIELDS* to anything but NULL. When providing data with a callback, you must transmit it using chunked transfer-encoding or you must set the size of the data with the *CURLOPT_POSTFIELDSIZE* option.

You can override the default POST Content-Type: header by setting your own with *CURLOPT_HTTPHEADER*.

Using POST with HTTP 1.1 implies the use of a "Expect: 100-continue" header. You can disable this header with *CURLOPT_HTTPHEADER* as usual.

If you use POST to a HTTP 1.1 server, you can send data without knowing the size before starting the POST if you use chunked encoding. You enable this by adding a header like "Transfer-Encoding: chunked" with *CURLOPT_HTTPHEADER*. With HTTP 1.0 or without chunked transfer, you must specify the size in the request.

When setting *CURLOPT_POST* to a non-zero value, it will automatically set *CURLOPT_NOBODY* to 0 (since 7.14.1).

If you issue a POST request and then want to make a HEAD or GET using the same re-used handle, you must explictly set the new request type using *CURLOPT_NOBODY* or *CURLOPT_HTTPGET* or similar.

CURLOPT_POSTFIELDS

Pass a char * as parameter, which should be the full data to post in an HTTP POST operation. You must make sure that the data is formatted the way you want the server to receive it. libcurl will not convert or encode it for you. Most web servers will assume this data to be url-encoded. Take note.

This POST is a normal application/x-www-form-urlencoded kind (and libcurl will set that Content-Type by default when this option is used), which is the most commonly used one by HTML forms. See also the *CURLOPT_POST*. Using *CURLOPT_POSTFIELDS* implies *CURLOPT_POST*.

Using POST with HTTP 1.1 implies the use of a "Expect: 100-continue" header. You can disable this header with *CURLOPT_HTTPHEADER* as usual.

To make multipart/formdata posts (aka rfc1867-posts), check out the *CURLOPT_HTTPPOST* option.

CURLOPT_POSTFIELDSIZE

If you want to post data to the server without letting libcurl do a strlen() to measure the data size, this option must be used. When this option is used you can post fully binary data, which otherwise is likely to fail. If this size is set to -1, the library will use strlen() to get the size.

CURLOPT_POSTFIELDSIZE_LARGE

Pass a curl_off_t as parameter. Use this to set the size of the *CURLOPT_POSTFIELDS* data to prevent libcurl from doing strlen() on the data to figure out the size. This is the large file version of the *CURLOPT_POSTFIELDSIZE* option. (Added in 7.11.1)

CURLOPT_HTTPPOST

Tells libcurl you want a multipart/formdata HTTP POST to be made and you instruct what data to pass on to the server. Pass a pointer to a linked list of curl_httppost structs as parameter. . The easiest way to create such a list, is to use *curl_formadd(3)* as documented. The data in this list must remain intact until you close this curl handle again with *curl_easy_cleanup(3)*.

Using POST with HTTP 1.1 implies the use of a "Expect: 100-continue" header. You can disable this header with *CURLOPT_HTTPHEADER* as usual.

When setting *CURLOPT_HTTPPOST*, it will automatically set *CURLOPT_NOBODY* to 0 (since 7.14.1).

CURLOPT_REFERER

Pass a pointer to a zero terminated string as parameter. It will be used to set the Referer: header in the http request sent to the remote server. This can be used to fool servers or scripts. You can also set any custom header with *CURLOPT_HTTPHEADER*.

CURLOPT_USERAGENT

Pass a pointer to a zero terminated string as parameter. It will be used to set the User-Agent: header in the http request sent to the remote server. This can be used to fool servers or scripts. You can also set any custom header with *CURLOPT_HTTPHEADER*.

CURLOPT_HTTPHEADER

Pass a pointer to a linked list of HTTP headers to pass to the server in your HTTP request. The linked list should be a fully valid list of **struct curl_slist** structs properly filled in. Use *curl_slist_append(3)* to create the list and *curl_slist_free_all(3)* to clean up an entire list. If you add a header that is otherwise generated and used by libcurl internally, your added one will be used instead. If you add a header with no contents as in 'Accept:' (no data on the right side of the colon), the internally used header will get disabled. Thus, using this option you can add new headers, replace internal headers and remove internal headers. To add a header with no contents, make the contents be two quotes: "". The headers included in the linked list must not be CRLF-terminated, because curl adds CRLF after each header item. Failure to comply with this will result in

strange bugs because the server will most likely ignore part of the headers you specified.

The first line in a request (containing the method, usually a GET or POST) is not a header and cannot be replaced using this option. Only the lines following the request-line are headers. Adding this method line in this list of headers will only cause your request to send an invalid header.

Pass a NULL to this to reset back to no custom headers.

The most commonly replaced headers have "shortcuts" in the options *CURLOPT_COOKIE*, *CURLOPT_USERAGENT* and *CURLOPT_REFERER*.

CURLOPT_HTTP200ALIASES

Pass a pointer to a linked list of aliases to be treated as valid HTTP 200 responses. Some servers respond with a custom header response line. For example, IceCast servers respond with "ICY 200 OK". By including this string in your list of aliases, the response will be treated as a valid HTTP header line such as "HTTP/1.0 200 OK". (Added in 7.10.3)

The linked list should be a fully valid list of struct curl_slist structs, and be properly filled in. Use *curl_slist_append(3)* to create the list and *curl_slist_free_all(3)* to clean up an entire list.

The alias itself is not parsed for any version strings. So if your alias is "MYHTTP/9.9", Libcurl will not treat the server as responding with HTTP version 9.9. Instead Libcurl will use the value set by option *CURLOPT_HTTP_VERSION*.

CURLOPT_COOKIE

Pass a pointer to a zero terminated string as parameter. It will be used to set a cookie in the http request. The format of the string should be NAME=CONTENTS, where NAME is the cookie name and CONTENTS is what the cookie should contain.

If you need to set multiple cookies, you need to set them all using a single option and thus you need to concatenate them all in one single string. Set multiple cookies in one string like this: "name1=content1; name2=content2;" etc.

Using this option multiple times will only make the latest string override the previously ones.

CURLOPT_COOKIEFILE

Pass a pointer to a zero terminated string as parameter. It should contain the name of your file holding cookie data to read. The cookie data may be in Netscape / Mozilla cookie data format or just regular HTTP-style headers dumped to a file.

Given an empty or non-existing file or by passing the empty string (""), this option will enable cookies for this curl handle, making it understand and parse received cookies and then use matching cookies in future request.

If you use this option multiple times, you just add more files to read. Subsequent files will add more cookies.

CURLOPT_COOKIEJAR

Pass a file name as char *, zero terminated. This will make libcurl write all internally known cookies to the specified file when *curl_easy_cleanup(3)* is called. If no cookies are known, no file will be created. Specify "-" to instead have the cookies written to stdout. Using this option also enables cookies for this session, so if you for example follow a location it will make matching cookies get sent accordingly.

If the cookie jar file can’t be created or written to (when the *curl_easy_cleanup(3)* is called), libcurl will not and cannot report an error for this. Using *CURLOPT_VERBOSE* or *CURLOPT_DEBUGFUNCTION* will get a warning to display, but that is the only visible feedback you

get about this possibly lethal situation.

CURLOPT_COOKIESESSION
: Pass a long set to non-zero to mark this as a new cookie "session". It will force libcurl to ignore all cookies it is about to load that are "session cookies" from the previous session. By default, libcurl always stores and loads all cookies, independent if they are session cookies are not. Session cookies are cookies without expiry date and they are meant to be alive and existing for this "session" only.

CURLOPT_COOKIELIST
: Pass a char * to a cookie string. Cookie can be either in Netscape / Mozilla format or just regular HTTP-style header (Set-Cookie: ...) format. If cURL cookie engine was not enabled it will enable its cookie engine. Passing a magic string "ALL" will erase all cookies known by cURL. (Added in 7.14.1) Passing the special string "SESS" will only erase all session cookies known by cURL. (Added in 7.15.4)

CURLOPT_HTTPGET
: Pass a long. If the long is non-zero, this forces the HTTP request to get back to GET. usable if a POST, HEAD, PUT or a custom request have been used previously using the same curl handle.

  When setting *CURLOPT_HTTPGET* to a non-zero value, it will automatically set *CURLOPT_NOBODY* to 0 (since 7.14.1).

CURLOPT_HTTP_VERSION
: Pass a long, set to one of the values described below. They force libcurl to use the specific HTTP versions. This is not sensible to do unless you have a good reason.

  CURL_HTTP_VERSION_NONE
  : We don't care about what version the library uses. libcurl will use whatever it thinks fit.

  CURL_HTTP_VERSION_1_0
  : Enforce HTTP 1.0 requests.

  CURL_HTTP_VERSION_1_1
  : Enforce HTTP 1.1 requests.

  CURLOPT_IGNORE_CONTENT_LENGTH
  : Ignore the Content-Length header. This is useful for Apache 1.x (and similar servers) which will report incorrect content length for files over 2 gigabytes. If this option is used, curl will not be able to accurately report progress, and will simply stop the download when the server ends the connection. (added in 7.14.1)

FTP OPTIONS

CURLOPT_FTPPORT
: Pass a pointer to a zero terminated string as parameter. It will be used to get the IP address to use for the ftp PORT instruction. The PORT instruction tells the remote server to connect to our specified IP address. The string may be a plain IP address, a host name, an network interface name (under Unix) or just a '-' letter to let the library use your systems default IP address. Default FTP operations are passive, and thus won't use PORT.

  You disable PORT again and go back to using the passive version by setting this option to NULL.

CURLOPT_QUOTE
: Pass a pointer to a linked list of FTP commands to pass to the server prior to your ftp request. This will be done before any other FTP commands are issued (even before the CWD command). The linked list should be a fully valid list of to append strings (commands) to the list, and clear the entire list afterwards with *curl_slist_free_all(3)*. Disable this operation again by setting a NULL to this option.

CURLOPT_POSTQUOTE

Pass a pointer to a linked list of FTP commands to pass to the server after your ftp transfer request. The linked list should be a fully valid list of struct curl_slist structs properly filled in as described for *CURLOPT_QUOTE*. Disable this operation again by setting a NULL to this option.

CURLOPT_PREQUOTE

Pass a pointer to a linked list of FTP commands to pass to the server after the transfer type is set. The linked list should be a fully valid list of struct curl_slist structs properly filled in as described for *CURLOPT_QUOTE*. Disable this operation again by setting a NULL to this option.

CURLOPT_FTPLISTONLY

A non-zero parameter tells the library to just list the names of an ftp directory, instead of doing a full directory listing that would include file sizes, dates etc.

This causes an FTP NLST command to be sent. Beware that some FTP servers list only files in their response to NLST; they might not include subdirectories and symbolic links.

CURLOPT_FTPAPPEND

A non-zero parameter tells the library to append to the remote file instead of overwrite it. This is only useful when uploading to an ftp site.

CURLOPT_FTP_USE_EPRT

Pass a long. If the value is non-zero, it tells curl to use the EPRT (and LPRT) command when doing active FTP downloads (which is enabled by *CURLOPT_FTPPORT*). Using EPRT means that it will first attempt to use EPRT and then LPRT before using PORT, but if you pass FALSE (zero) to this option, it will not try using EPRT or LPRT, only plain PORT. (Added in 7.10.5)

If the server is an IPv6 host, this option will have no effect as of 7.12.3.

CURLOPT_FTP_USE_EPSV

Pass a long. If the value is non-zero, it tells curl to use the EPSV command when doing passive FTP downloads (which it always does by default). Using EPSV means that it will first attempt to use EPSV before using PASV, but if you pass FALSE (zero) to this option, it will not try using EPSV, only plain PASV.

If the server is an IPv6 host, this option will have no effect as of 7.12.3.

CURLOPT_FTP_CREATE_MISSING_DIRS

Pass a long. If the value is non-zero, curl will attempt to create any remote directory that it fails to CWD into. CWD is the command that changes working directory. (Added in 7.10.7)

CURLOPT_FTP_RESPONSE_TIMEOUT

Pass a long. Causes curl to set a timeout period (in seconds) on the amount of time that the server is allowed to take in order to generate a response message for a command before the session is considered hung. While curl is waiting for a response, this value overrides *CURLOPT_TIMEOUT*. It is recommended that if used in conjunction with *CURLOPT_TIMEOUT*, you set *CURLOPT_FTP_RESPONSE_TIMEOUT* to a value smaller than *CURLOPT_TIMEOUT*. (Added in 7.10.8)

CURLOPT_FTP_SKIP_PASV_IP

Pass a long. If set to a non-zero value, it instructs libcurl to not use the IP address the server suggests in its 227-response to libcurl's PASV command when libcurl connects the data connection. Instead libcurl will re-use the same IP address it already uses for the control connection. But it will use the port number from the 227-response. (Added in 7.14.2)

This option has no effect if PORT, EPRT or EPSV is used instead of PASV.

CURLOPT_FTP_SSL

Pass a long using one of the values from below, to make libcurl use your desired level of SSL for the ftp transfer. (Added in 7.11.0)

CURLFTPSSL_NONE
Don’t attempt to use SSL.

CURLFTPSSL_TRY
Try using SSL, proceed as normal otherwise.

CURLFTPSSL_CONTROL
Require SSL for the control connection or fail with *CURLE_FTP_SSL_FAILED*.

CURLFTPSSL_ALL
Require SSL for all communication or fail with *CURLE_FTP_SSL_FAILED*.

CURLOPT_FTPSSLAUTH
Pass a long using one of the values from below, to alter how libcurl issues "AUTH TLS" or "AUTH SSL" when FTP over SSL is activated (see *CURLOPT_FTP_SSL*). (Added in 7.12.2)

CURLFTPAUTH_DEFAULT
Allow libcurl to decide

CURLFTPAUTH_SSL
Try "AUTH SSL" first, and only if that fails try "AUTH TLS"

CURLFTPAUTH_TLS
Try "AUTH TLS" first, and only if that fails try "AUTH SSL"

CURLOPT_SOURCE_URL
When set, it enables a FTP third party transfer, using the set URL as source, while *CURLOPT_URL* is the target.

CURLOPT_SOURCE_USERPWD
Set "username:password" to use for the source connection when doing FTP third party transfers.

CURLOPT_SOURCE_QUOTE
Exactly like *CURLOPT_QUOTE*, but for the source host.

CURLOPT_SOURCE_PREQUOTE
Exactly like *CURLOPT_PREQUOTE*, but for the source host.

CURLOPT_SOURCE_POSTQUOTE
Exactly like *CURLOPT_POSTQUOTE*, but for the source host.

CURLOPT_FTP_ACCOUNT
Pass a pointer to a zero-terminated string (or NULL to disable). When an FTP server asks for "account data" after user name and password has been provided, this data is sent off using the ACCT command. (Added in 7.13.0)

CURLOPT_FTP_FILEMETHOD
Pass a long that should have one of the following values. This option controls what method libcurl should use to reach a file on a FTP(S) server. The argument should be one of the following alternatives:

CURLFTPMETHOD_MULTICWD
libcurl does a single CWD operation for each path part in the given URL. For deep hierarchies this means very many commands. This is how RFC1738 says it should be done. This is the default but the slowest behavior.

CURLFTPMETHOD_NOCWD
libcurl does no CWD at all. libcurl will do SIZE, RETR, STOR etc and give a full path to the server for all these commands. This is the fastest behavior.

CURLFTPMETHOD_SINGLECWD
libcurl does one CWD with the full target directory and then operates on the file "normally" (like in the multicwd case). This is somewhat more standards compliant than ’nocwd’ but without the full penalty of ’multicwd’.

PROTOCOL OPTIONS

CURLOPT_TRANSFERTEXT

A non-zero parameter tells the library to use ASCII mode for ftp transfers, instead of the default binary transfer. For win32 systems it does not set the stdout to binary mode. This option can be usable when transferring text data between systems with different views on certain characters, such as newlines or similar.

libcurl does not do a complete ASCII conversion when doing ASCII transfers over FTP. This is a known limitation/flaw that nobody has rectified. libcurl simply sets the mode to ascii and performs a standard transfer.

CURLOPT_CRLF

Convert Unix newlines to CRLF newlines on transfers.

CURLOPT_RANGE

Pass a char * as parameter, which should contain the specified range you want. It should be in the format "X-Y", where X or Y may be left out. HTTP transfers also support several intervals, separated with commas as in *"X-Y,N-M"*. Using this kind of multiple intervals will cause the HTTP server to send the response document in pieces (using standard MIME separation techniques). Pass a NULL to this option to disable the use of ranges.

CURLOPT_RESUME_FROM

Pass a long as parameter. It contains the offset in number of bytes that you want the transfer to start from. Set this option to 0 to make the transfer start from the beginning (effectively disabling resume).

CURLOPT_RESUME_FROM_LARGE

Pass a curl_off_t as parameter. It contains the offset in number of bytes that you want the transfer to start from. (Added in 7.11.0)

CURLOPT_CUSTOMREQUEST

Pass a pointer to a zero terminated string as parameter. It will be user instead of GET or HEAD when doing an HTTP request, or instead of LIST or NLST when doing an ftp directory listing. This is useful for doing DELETE or other more or less obscure HTTP requests. Don’t do this at will, make sure your server supports the command first.

Restore to the internal default by setting this to NULL.

Many people have wrongly used this option to replace the entire request with their own, including multiple headers and POST contents. While that might work in many cases, it will cause libcurl to send invalid requests and it could possibly confuse the remote server badly. Use *CURLOPT_POST* and *CURLOPT_POSTFIELDS* to set POST data. Use *CURLOPT_HTTPHEADER* to replace or extend the set of headers sent by libcurl. Use *CURLOPT_HTTP_VERSION* to change HTTP version.

CURLOPT_FILETIME

Pass a long. If it is a non-zero value, libcurl will attempt to get the modification date of the remote document in this operation. This requires that the remote server sends the time or replies to a time querying command. The *curl_easy_getinfo(3)* function with the *CURLINFO_FILETIME* argument can be used after a transfer to extract the received time (if any).

CURLOPT_NOBODY

A non-zero parameter tells the library to not include the body-part in the output. This is only relevant for protocols that have separate header and body parts. On HTTP(S) servers, this will make libcurl do a HEAD request.

To change request to GET, you should use *CURLOPT_HTTPGET*. Change request to POST with *CURLOPT_POST* etc.

CURLOPT_INFILESIZE

When uploading a file to a remote site, this option should be used to tell libcurl what the expected size of the infile is. This value should be passed as a long. See also *CURLOPT_INFILESIZE_LARGE*.

CURLOPT_INFILESIZE_LARGE

When uploading a file to a remote site, this option should be used to tell libcurl what the expected size of the infile is. This value should be passed as a curl_off_t. (Added in 7.11.0)

CURLOPT_UPLOAD

A non-zero parameter tells the library to prepare for an upload. The *CURLOPT_READDATA* and *CURLOPT_INFILESIZEE* or *CURLOPT_INFILESIZE_LARGE* are also interesting for uploads. If the protocol is HTTP, uploading means using the PUT request unless you tell libcurl otherwise.

Using PUT with HTTP 1.1 implies the use of a "Expect: 100-continue" header. You can disable this header with *CURLOPT_HTTPHEADER* as usual.

If you use PUT to a HTTP 1.1 server, you can upload data without knowing the size before starting the transfer if you use chunked encoding. You enable this by adding a header like "Transfer-Encoding: chunked" with *CURLOPT_HTTPHEADER*. With HTTP 1.0 or without chunked transfer, you must specify the size.

CURLOPT_MAXFILESIZE

Pass a long as parameter. This allows you to specify the maximum size (in bytes) of a file to download. If the file requested is larger than this value, the transfer will not start and CURLE_FILESIZE_EXCEEDED will be returned.

The file size is not always known prior to download, and for such files this option has no effect even if the file transfer ends up being larger than this given limit. This concerns both FTP and HTTP transfers.

CURLOPT_MAXFILESIZE_LARGE

Pass a curl_off_t as parameter. This allows you to specify the maximum size (in bytes) of a file to download. If the file requested is larger than this value, the transfer will not start and *CURLE_FILESIZE_EXCEEDED* will be returned. (Added in 7.11.0)

The file size is not always known prior to download, and for such files this option has no effect even if the file transfer ends up being larger than this given limit. This concerns both FTP and HTTP transfers.

CURLOPT_TIMECONDITION

Pass a long as parameter. This defines how the *CURLOPT_TIMEVALUE* time value is treated. You can set this parameter to *CURL_TIMECOND_IFMODSINCE* or *CURL_TIMECOND_IFUNMODSINCE*. This feature applies to HTTP and FTP.

The last modification time of a file is not always known and in such instances this feature will have no effect even if the given time condition would have not been met.

CURLOPT_TIMEVALUE

Pass a long as parameter. This should be the time in seconds since 1 jan 1970, and the time will be used in a condition as specified with *CURLOPT_TIMECONDITION*.

CONNECTION OPTIONS

CURLOPT_TIMEOUT

Pass a long as parameter containing the maximum time in seconds that you allow the libcurl transfer operation to take. Normally, name lookups can take a considerable time and limiting operations to less than a few minutes risk aborting perfectly normal operations. This option will cause curl to use the SIGALRM to enable time-outing system calls.

In unix-like systems, this might cause signals to be used unless *CURLOPT_NOSIGNAL* is set.

CURLOPT_LOW_SPEED_LIMIT
: Pass a long as parameter. It contains the transfer speed in bytes per second that the transfer should be below during *CURLOPT_LOW_SPEED_TIME* seconds for the library to consider it too slow and abort.

CURLOPT_LOW_SPEED_TIME
: Pass a long as parameter. It contains the time in seconds that the transfer should be below the *CURLOPT_LOW_SPEED_LIMIT* for the library to consider it too slow and abort.

CURLOPT_MAXCONNECTS
: Pass a long. The set number will be the persistent connection cache size. The set amount will be the maximum amount of simultaneously open connections that libcurl may cache. Default is 5, and there isn't much point in changing this value unless you are perfectly aware of how this work and changes libcurl's behaviour. This concerns connection using any of the protocols that support persistent connections.

  When reaching the maximum limit, curl uses the *CURLOPT_CLOSEPOLICY* to figure out which of the existing connections to close to prevent the number of open connections to increase.

  If you already have performed transfers with this curl handle, setting a smaller MAXCONNECTS than before may cause open connections to get closed unnecessarily.

CURLOPT_CLOSEPOLICY
: Pass a long. This option sets what policy libcurl should use when the connection cache is filled and one of the open connections has to be closed to make room for a new connection. This must be one of the CURLCLOSEPOLICY_* defines. Use *CURLCLOSEPOLICY_LEAST_RECENTLY_USED* to make libcurl close the connection that was least recently used, that connection is also least likely to be capable of re-use. Use *CURLCLOSEPOLICY_OLDEST* to make libcurl close the oldest connection, the one that was created first among the ones in the connection cache. The other close policies are not support yet.

CURLOPT_FRESH_CONNECT
: Pass a long. Set to non-zero to make the next transfer use a new (fresh) connection by force. If the connection cache is full before this connection, one of the existing connections will be closed as according to the selected or default policy. This option should be used with caution and only if you understand what it does. Set this to 0 to have libcurl attempt re-using an existing connection (default behavior).

CURLOPT_FORBID_REUSE
: Pass a long. Set to non-zero to make the next transfer explicitly close the connection when done. Normally, libcurl keep all connections alive when done with one transfer in case there comes a succeeding one that can re-use them. This option should be used with caution and only if you understand what it does. Set to 0 to have libcurl keep the connection open for possibly later re-use (default behavior).

CURLOPT_CONNECTTIMEOUT
: Pass a long. It should contain the maximum time in seconds that you allow the connection to the server to take. This only limits the connection phase, once it has connected, this option is of no more use. Set to zero to disable connection timeout (it will then only timeout on the system's internal timeouts). See also the *CURLOPT_TIMEOUT* option.

  In unix-like systems, this might cause signals to be used unless *CURLOPT_NOSIGNAL* is set.

CURLOPT_IPRESOLVE
: Allows an application to select what kind of IP addresses to use when resolving host names. This is only interesting when using host names that resolve addresses using more than one version of IP. The allowed values are:

CURL_IPRESOLVE_WHATEVER
Default, resolves addresses to all IP versions that your system allows.

CURL_IPRESOLVE_V4
Resolve to ipv4 addresses.

CURL_IPRESOLVE_V6
Resolve to ipv6 addresses.

CURLOPT_CONNECT_ONLY

Pass a long. A non-zero parameter tells the library to perform any required proxy authentication and connection setup, but no data transfer.

This option is useful with the *CURLINFO_LASTSOCKET* option to *curl_easy_getinfo(3)*. The library can set up the connection and then the application can obtain the most recently used socket for special data transfers. (Added in 7.15.2)

SSL and SECURITY OPTIONS

CURLOPT_SSLCERT
Pass a pointer to a zero terminated string as parameter. The string should be the file name of your certificate. The default format is "PEM" and can be changed with *CURLOPT_SSLCERTTYPE*.

CURLOPT_SSLCERTTYPE
Pass a pointer to a zero terminated string as parameter. The string should be the format of your certificate. Supported formats are "PEM" and "DER". (Added in 7.9.3)

CURLOPT_SSLCERTPASSWD
Pass a pointer to a zero terminated string as parameter. It will be used as the password required to use the *CURLOPT_SSLCERT* certificate.

This option is replaced by *CURLOPT_SSLKEYPASSWD* and should only be used for backward compatibility. You never needed a pass phrase to load a certificate but you need one to load your private key.

CURLOPT_SSLKEY
Pass a pointer to a zero terminated string as parameter. The string should be the file name of your private key. The default format is "PEM" and can be changed with *CURLOPT_SSLKEYTYPE*.

CURLOPT_SSLKEYTYPE
Pass a pointer to a zero terminated string as parameter. The string should be the format of your private key. Supported formats are "PEM", "DER" and "ENG".

The format "ENG" enables you to load the private key from a crypto engine. In this case *CURLOPT_SSLKEY* is used as an identifier passed to the engine. You have to set the crypto engine with *CURLOPT_SSLENGINE*. "DER" format key file currently does not work because of a bug in OpenSSL.

CURLOPT_SSLKEYPASSWD
Pass a pointer to a zero terminated string as parameter. It will be used as the password required to use the *CURLOPT_SSLKEY* private key.

CURLOPT_SSLENGINE
Pass a pointer to a zero terminated string as parameter. It will be used as the identifier for the crypto engine you want to use for your private key.

If the crypto device cannot be loaded, *CURLE_SSL_ENGINE_NOTFOUND* is returned.

CURLOPT_SSLENGINE_DEFAULT
Sets the actual crypto engine as the default for (asymmetric) crypto operations.

If the crypto device cannot be set, *CURLE_SSL_ENGINE_SETFAILED* is returned.

CURLOPT_SSLVERSION

Pass a long as parameter to control what version of SSL/TLS to attempt to use. The available options are:

CURL_SSLVERSION_DEFAULT

The default action. When libcurl built with OpenSSL, this will attempt to figure out the remote SSL protocol version. Unfortunately there are a lot of ancient and broken servers in use which cannot handle this technique and will fail to connect. When libcurl is built with GnuTLS, this will mean SSLv3.

CURL_SSLVERSION_TLSv1

Force TLSv1

CURL_SSLVERSION_SSLv2

Force SSLv2

CURL_SSLVERSION_SSLv3

Force SSLv3

CURLOPT_SSL_VERIFYPEER

Pass a long as parameter.

This option determines whether curl verifies the authenticity of the peer's certificate. A nonzero value means curl verifies; zero means it doesn't. The default is nonzero, but before 7.10, it was zero.

When negotiating an SSL connection, the server sends a certificate indicating its identity. Curl verifies whether the certificate is authentic, i.e. that you can trust that the server is who the certificate says it is. This trust is based on a chain of digital signatures, rooted in certification authority (CA) certificates you supply. As of 7.10, curl installs a default bundle of CA certificates and you can specify alternate certificates with the *CURLOPT_CAINFO* option or the *CURLOPT_CAPATH* option.

When *CURLOPT_SSL_VERIFYPEER* is nonzero, and the verification fails to prove that the certificate is authentic, the connection fails. When the option is zero, the connection succeeds regardless.

Authenticating the certificate is not by itself very useful. You typically want to ensure that the server, as authentically identified by its certificate, is the server you mean to be talking to. Use *CURLOPT_SSL_VERIFYHOST* to control that.

CURLOPT_CAINFO

Pass a char * to a zero terminated string naming a file holding one or more certificates to verify the peer with. This makes sense only when used in combination with the *CURLOPT_SSL_VERIFYPEER* option. If *CURLOPT_SSL_VERIFYPEER* is zero, *CURLOPT_CAINFO* need not even indicate an accessible file.

Note that option is by default set to the system path where libcurl's cacert bundle is assumed to be stored, as established at build time.

CURLOPT_CAPATH

Pass a char * to a zero terminated string naming a directory holding multiple CA certificates to verify the peer with. The certificate directory must be prepared using the openssl c_rehash utility. This makes sense only when used in combination with the *CURLOPT_SSL_VERIFYPEER* option. If *CURLOPT_SSL_VERIFYPEER* is zero, *CURLOPT_CAPATH* need not even indicate an accessible path. The *CURLOPT_CAPATH* function apparently does not work in Windows due to some limitation in openssl. (Added in 7.9.8)

CURLOPT_RANDOM_FILE

Pass a char * to a zero terminated file name. The file will be used to read from to seed the random engine for SSL. The more random the specified file is, the more secure the SSL connection will become.

CURLOPT_EGDSOCKET

Pass a char * to the zero terminated path name to the Entropy Gathering Daemon socket. It will be used to seed the random engine for SSL.

CURLOPT_SSL_VERIFYHOST

Pass a long as parameter.

This option determines whether libcurl verifies that the server cert is for the server it is known as.

When negotiating an SSL connection, the server sends a certificate indicating its identity.

When *CURLOPT_SSL_VERIFYHOST* is 2, that certificate must indicate that the server is the server to which you meant to connect, or the connection fails.

Curl considers the server the intended one when the Common Name field or a Subject Alternate Name field in the certificate matches the host name in the URL to which you told Curl to connect.

When the value is 1, the certificate must contain a Common Name field, but it doesn't matter what name it says. (This is not ordinarily a useful setting).

When the value is 0, the connection succeeds regardless of the names in the certificate.

The default, since 7.10, is 2.

The checking this option controls is of the identity that the server *claims*. The server could be lying. To control lying, see *CURLOPT_SSL_VERIFYPEER*.

CURLOPT_SSL_CIPHER_LIST

Pass a char *, pointing to a zero terminated string holding the list of ciphers to use for the SSL connection. The list must be syntactically correct, it consists of one or more cipher strings separated by colons. Commas or spaces are also acceptable separators but colons are normally used, , − and + can be used as operators. Valid examples of cipher lists include 'RC4-SHA', ´SHA1+DES´, 'TLSv1' and 'DEFAULT'. The default list is normally set when you compile OpenSSL.

You'll find more details about cipher lists on this URL: *http://www.openssl.org/docs/apps/ciphers.html*

CURLOPT_KRB4LEVEL

Pass a char * as parameter. Set the krb4 security level, this also enables krb4 awareness. This is a string, 'clear', 'safe', 'confidential' or 'private'. If the string is set but doesn't match one of these, 'private' will be used. Set the string to NULL to disable kerberos4. The kerberos support only works for FTP.

OTHER OPTIONS

CURLOPT_PRIVATE

Pass a char * as parameter, pointing to data that should be associated with this curl handle. The pointer can subsequently be retrieved using *curl_easy_getinfo(3)* with the CURLINFO_PRIVATE option. libcurl itself does nothing with this data. (Added in 7.10.3)

CURLOPT_SHARE

Pass a share handle as a parameter. The share handle must have been created by a previous call to *curl_share_init(3)*. Setting this option, will make this curl handle use the data from the shared handle instead of keeping the data to itself. This enables several curl handles to share data. If the curl

handles are used simultaneously, you **MUST** use the locking methods in the share handle. See *curl_share_setopt(3)* for details.

TELNET OPTIONS

CURLOPT_TELNETOPTIONS

Provide a pointer to a curl_slist with variables to pass to the telnet negotiations. The variables should be in the format <option=value>. libcurl supports the options 'TTYPE', 'XDISPLOC' and 'NEW_ENV'. See the TELNET standard for details.

RETURN VALUE

CURLE_OK (zero) means that the option was set properly, non-zero means an error occurred as *<curl/curl.h>* defines. See the *libcurl-errors(3)* man page for the full list with descriptions.

If you try to set an option that libcurl doesn't know about, perhaps because the library is too old to support it or the option was removed in a recent version, this function will return *CURLE_FAILED_INIT*.

SEE ALSO

curl_easy_init(3), **curl_easy_cleanup**(3), **curl_easy_reset**(3),