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, are copied by the library; thus the string storage associated to the pointer argument may be overwritten after curl_easy_setopt() returns. Exceptions to this rule are described in the option details below.

Before version 7.17.0, strings were not copied. Instead the user was forced keep them available until libcurl no longer needed them.

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 1 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 parameter set to 1 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

Pass a long. If set to 1, it tells the library to shut off the progress meter completely. It will also present the *CURLOPT_PROGRESSFUNCTION* from getting called.

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

CURLOPT_NOSIGNAL

Pass a long. If it is 1, 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)

If this option is set and libcurl has been built with the standard name resolver, timeouts will not occur while the name resolve takes place. Consider building libcurl with c-ares support to enable asynchronous DNS lookups, which enables nice timeouts for name resolves without signals.

Setting *CURLOPT_NOSIGNAL* to 1 makes libcurl NOT ask the system to ignore SIGPIPE signals, which otherwise are sent by the system when trying to send data to a socket which is closed in the other end. libcurl makes an effort to never cause such SIGPIPEs to trigger, but some operating systems have no way to avoid them and even on those that have there are some corner cases when they may still happen, contrary to our desire. In addition, using *CURLAUTH_NTLM_WB* authentication could cause a SIGCHLD signal to be raised.

CURLOPT_WILDCARDMATCH

Set this option to 1 if you want to transfer multiple files according to a file name pattern. The pattern can be specified as part of the *CURLOPT_URL* option, using an fnmatch-like pattern (Shell Pattern Matching) in the last part of URL (file name).

By default, libcurl uses its internal wildcard matching implementation. You can provide your own matching function by the *CURLOPT_FNMATCH_FUNCTION* option.

This feature is only supported by the FTP download for now.

A brief introduction of its syntax follows:

* - ASTERISK

ftp://example.com/some/path/***.txt** (for all txt's from the root directory)

? - QUESTION MARK

Question mark matches any (exactly one) character.

ftp://example.com/some/path/**photo?.jpeg**

[- BRACKET EXPRESSION

The left bracket opens a bracket expression. The question mark and asterisk have no special meaning in a bracket expression. Each bracket expression ends by the right bracket and matches exactly one character. Some examples follow:

[a-zA-Z0-9] or **[f-gF-G]** - character interval

[abc] - character enumeration

[^abc] or **[!abc]** - negation

[[:*name***:]]** class expression. Supported classes are **alnum**,**lower**, **space**, **alpha**, **digit**, **print**, **upper**, **blank**, **graph**, **xdigit**.

[][-!^] - special case - matches only '-', ']', '[', '!' or '^'. These characters have no special purpose.

[\[\]\\] - escape syntax. Matches '[', ']' or '\'.

Using the rules above, a file name pattern can be constructed:

ftp://example.com/some/path/**[a-z[:upper:]\\].jpeg**

(This was added in 7.21.0)

CALLBACK OPTIONS

CURLOPT_WRITEFUNCTION

Function pointer that should match the following prototype: **size_t function(char *ptr, size_t size, size_t nmemb, void *userdata);** 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. This will abort the transfer and return *CURLE_WRITE_ERROR*.

From 7.18.0, the function can return CURL_WRITEFUNC_PAUSE which then will cause writing to this connection to become paused. See *curl_easy_pause(3)* for further details.

This function may be called with zero bytes data if the transferred 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 *userdata* 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 body data that can be passed to the write callback is defined in the curl.h header file: CURL_MAX_WRITE_SIZE (the usual default is 16K). If you however have *CURLOPT_HEADER* set, which sends header data to the write callback, you can get up to *CURL_MAX_HTTP_HEADER* bytes of header data passed into it. This usually means 100K.

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 *userdata);** 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 said 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)

From 7.18.0, the function can return CURL_READFUNC_PAUSE which then will cause reading from this connection to become paused. See *curl_easy_pause(3)* for further details.

Bugs: when doing TFTP uploads, you must return the exact amount of data that the callback wants, or it will be considered the final packet by the server end and the transfer will end there.

If you set this callback pointer to NULL, or don't set it at all, the default internal read function will be used. It is doing an fread() on the FILE * userdata 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 was also known by 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. (Option added in 7.12.3).

Use *CURLOPT_SEEKFUNCTION* instead to provide seeking!

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_SEEKFUNCTION

Function pointer that should match the following prototype: *int function(void *instream, curl_off_t offset, int origin);* This function gets called by libcurl to seek to a certain position in the input stream and can be used to fast forward a file in a resumed upload (instead of reading all uploaded bytes with the normal read function/callback). It is also called to rewind a stream when doing a HTTP PUT or POST with a multi-pass authentication method. The function shall work like "fseek" or "lseek" and accepted SEEK_SET, SEEK_CUR and SEEK_END as argument for origin, although (in 7.18.0) libcurl only passes SEEK_SET. The callback must return 0 (CURL_SEEKFUNC_OK) on success, 1 (CURL_SEEKFUNC_FAIL) to cause the upload operation to fail or 2 (CURL_SEEKFUNC_CANTSEEK) to indicate that while the seek failed, libcurl is free to work around the problem if possible. The latter can sometimes be done by instead reading from the input or similar.

If you forward the input arguments directly to "fseek" or "lseek", note that the data type for *offset* is not the same as defined for curl_off_t on many systems! (Option added in 7.18.0)

CURLOPT_SEEKDATA

Data pointer to pass to the file seek function. If you use the *CURLOPT_SEEKFUNCTION* option, this is the pointer you'll get as input. If you don't specify a seek callback, NULL is passed. (Option added in 7.18.0)

CURLOPT_SOCKOPTFUNCTION

Function pointer that should match the *curl_sockopt_callback* prototype found in <*curl/curl.h*>. This function gets called by libcurl after the socket() call but before the connect() call. The callback's *purpose* argument identifies the exact purpose for this particular socket, and currently only one value is supported: *CURLSOCKTYPE_IPCXN* for the primary connection (meaning the control connection in the FTP case). Future versions of libcurl may support more purposes. It passes the newly created socket descriptor so additional setsockopt() calls can be done at the user's discretion. Return 0 (zero) from the callback on success. Return 1 from the callback function to signal an unrecoverable error to the library and it will close the socket and return *CURLE_COULDNT_CONNECT*. (Option added in 7.16.0)

Added in 7.21.5, the callback function may return *CURL_SOCKOPT_ALREADY_CONNECTED*, which tells libcurl that the socket is in fact already connected and then libcurl will not attempt to connect it.

CURLOPT_SOCKOPTDATA

Pass a pointer that will be untouched by libcurl and passed as the first argument in the sockopt callback set with *CURLOPT_SOCKOPTFUNCTION*. (Option added in 7.16.0)

CURLOPT_OPENSOCKETFUNCTION

Function pointer that should match the *curl_opensocket_callback* prototype found in *<curl/curl.h>*. This function gets called by libcurl instead of the *socket(2)* call. The callback's *purpose* argument identifies the exact purpose for this particular socket: *CURLSOCKTYPE_IPCXN* is for IP based connections. Future versions of libcurl may support more purposes. It passes the resolved peer address as a *address* argument so the callback can modify the address or refuse to connect at all. The callback function should return the socket or *CURL_SOCKET_BAD* in case no connection should be established or any error detected. Any additional *setsockopt(2)* calls can be done on the socket at the user's discretion. *CURL_SOCKET_BAD* return value from the callback function will signal an unrecoverable error to the library and it will return *CURLE_COULDNT_CONNECT*. This return code can be used for IP address blacklisting. The default behavior is:

```
return socket(addr->family, addr->socktype, addr->protocol);
```

(Option added in 7.17.1.)

CURLOPT_OPENSOCKETDATA

Pass a pointer that will be untouched by libcurl and passed as the first argument in the opensocket callback set with *CURLOPT_OPENSOCKETFUNCTION*. (Option added in 7.17.1.)

CURLOPT_CLOSESOCKETFUNCTION

Function pointer that should match the *curl_closesocket_callback* prototype found in *<curl/curl.h>*. This function gets called by libcurl instead of the *close(3)* or *closesocket(3)* call when sockets are closed (not for any other file descriptors). This is pretty much the reverse to the *CURLOPT_OPENSOCKETFUNCTION* option. Return 0 to signal success and 1 if there was an error. (Option added in 7.21.7)

CURLOPT_CLOSESOCKETDATA

Pass a pointer that will be untouched by libcurl and passed as the first argument in the closesocket callback set with *CURLOPT_CLOSESOCKETFUNCTION*. (Option added in 7.21.7)

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 operation (roughly once per second or sooner) no matter if data is being transfered or not. Unknown/unused argument values passed 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.

CURLOPT_NOPROGRESS must be set to 0 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 *userdata);*. 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 is very easy 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 *userdata* is the one you set with the *CURLOPT_WRITEHEADER* option. The callback function must 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. This will abort the transfer and return *CURL_WRITE_ERROR*.

A complete HTTP header that is passed to this function can be up to *CURL_MAX_HTTP_HEADER* (100K) bytes.

If this option is not set, or if it is set to NULL, but *CURLOPT_HEADERDATA* (*CURLOPT_WRITEHEADER*) is set to anything but NULL, the function used to accept response data will be used instead. That is, it will be the function specified with *CURLOPT_WRITEFUNCTION*, or if it is not specified or NULL - the default, stream-writing function.

It's important to note that the callback will be invoked for the headers of all responses received after initiating a request and not just the final response. This includes all responses which occur during authentication negotiation. If you need to operate on only the headers from the final response, you will need to collect headers in the callback yourself and use HTTP status lines, for example, to delimit response boundaries.

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 regular response-headers mention what header(s) to expect in the trailer.

For non-HTTP protocols like FTP, POP3, IMAP and SMTP this function will get called with the server responses to the commands that libcurl sends.

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 *CURLOPT_WRITEFUNCTION* or *CURLOPT_HEADERFUNCTION* to take care of the writing, this must be a valid FILE * as the internal default will then be a plain fwrite(). 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
This option does only function for libcurl powered by OpenSSL. If libcurl was built against another SSL library, this functionality is absent.

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 a 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. For example, using this function allows you to use openssl callbacks to add additional validation code for certificates, and even to change the actual URI of a 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.

CURLOPT_INTERLEAVEFUNCTION

Function pointer that should match the following prototype: *size_t function(void *ptr, size_t size, size_t nmemb, void *userdata)*. This function gets called by libcurl as soon as it has received interleaved RTP data. This function gets called for each $ block and therefore contains exactly one upper-layer protocol unit (e.g. one RTP packet). Curl writes the interleaved header as well as the included data for each call. The first byte is always an ASCII dollar sign. The dollar sign is followed by a one byte channel identifier and then a 2 byte integer length in network byte order. See *RFC 2326 Section 10.12* for more information on how RTP interleaving behaves. If unset or set to NULL, curl will use the default write function.

Interleaved RTP poses some challeneges for the client application. Since the stream data is sharing the RTSP control connection, it is critical to service the RTP in a timely fashion. If the RTP data is not handled quickly, subsequent response processing may become unreasonably delayed and the connection may close. The application may use *CURL_RTSPREQ_RECEIVE* to service RTP data when no requests are desired. If the application makes a request, (e.g. *CURL_RTSPREQ_PAUSE*) then the response handler will process any pending RTP data before marking the request as finished. (Added in 7.20.0)

CURLOPT_INTERLEAVEDATA

This is the userdata pointer that will be passed to *CURLOPT_INTERLEAVEFUNCTION* when interleaved RTP data is received. (Added in 7.20.0)

CURLOPT_CHUNK_BGN_FUNCTION

Function pointer that should match the following prototype: **long function (const void *transfer_info, void *ptr, int remains)**. This function gets called by libcurl before a part of the stream is going to be transferred (if the transfer supports chunks).

This callback makes sense only when using the *CURLOPT_WILDCARDMATCH* option for now.

The target of transfer_info parameter is a "feature depended" structure. For the FTP wildcard download, the target is curl_fileinfo structure (see *curl/curl.h*). The parameter ptr is a pointer given by *CURLOPT_CHUNK_DATA*. The parameter remains contains number of chunks remaining per the transfer. If the feature is not available, the parameter has zero value.

Return *CURL_CHUNK_BGN_FUNC_OK* if everything is fine, *CURL_CHUNK_BGN_FUNC_SKIP* if you want to skip the concrete chunk or *CURL_CHUNK_BGN_FUNC_FAIL* to tell libcurl to stop if some error occurred. (This was added

in 7.21.0)

CURLOPT_CHUNK_END_FUNCTION

Function pointer that should match the following prototype: **long function(void *ptr)**. This function gets called by libcurl as soon as a part of the stream has been transferred (or skipped).

Return *CURL_CHUNK_END_FUNC_OK* if everything is fine or **CURL_CHUNK_END_FUNC_FAIL** to tell the lib to stop if some error occurred. (This was added in 7.21.0)

CURLOPT_CHUNK_DATA

Pass a pointer that will be untouched by libcurl and passed as the ptr argument to the *CURL_CHUNK_BGN_FUNTION* and *CURL_CHUNK_END_FUNTION*. (This was added in 7.21.0)

CURLOPT_FNMATCH_FUNCTION

Function pointer that should match **int function(void *ptr, const char *pattern, const char *string)** prototype (see *curl/curl.h*). It is used internally for the wildcard matching feature.

Return *CURL_FNMATCHFUNC_MATCH* if pattern matches the string, *CURL_FNMATCHFUNC_NOMATCH* if not or *CURL_FNMATCHFUNC_FAIL* if an error occurred. (This was added in 7.21.0)

CURLOPT_FNMATCH_DATA

Pass a pointer that will be untouched by libcurl and passed as the ptr argument to the *CURL_FNMATCH_FUNCTION*. (This was added in 7.21.0)

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. Although this argument is a 'char *', it does not describe an input string. Therefore the (probably undefined) contents of the buffer is NOT copied by the library. You should keep the associated storage available until libcurl no longer needs it. Failing to do so will cause very odd behavior or even crashes. libcurl will need it until you call *curl_easy_cleanup(3)* or you set the same option again to use a different pointer.

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 parameter set to 1 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.

This method is not fail-safe and there are occasions where non-successful response codes will slip through, especially when authentication is involved (response codes 401 and 407).

You might get some amounts of headers transferred before this situation is detected, like when a "100-continue" is received as a response to a POST/PUT and a 401 or 407 is received immediately afterwards.

NETWORK OPTIONS

CURLOPT_URL

The actual URL to deal with. The parameter should be a char * to a zero terminated string which must be URL-encoded in the following format:

scheme://host:port/path

For a greater explanation of the format please see RFC 3986 (http://curl.haxx.se/rfc/rfc3986.txt).

If the given URL lacks the scheme, or protocol, part ("http://" or "ftp://" etc), libcurl will attempt to resolve which protocol to use based on the given host mame. If the protocol is not supported, libcurl will return (*CURLE_UNSUPPORTED_PROTOCOL*) when you call *curl_easy_perform(3)* or *curl_multi_perform(3)*. Use *curl_version_info(3)* for detailed information on which protocols are supported.

The host part of the URL contains the address of the server that you want to connect to. This can be the fully qualified domain name of the server, the local network name of the machine on your network or the IP address of the server or machine represented by either an IPv4 or IPv6 address. For example:

http://www.example.com/

http://hostname/

http://192.168.0.1/

http://[2001:1890:1112:1::20]/

It is also possible to specify the user name and password as part of the host, for some protocols, when connecting to servers that require authentication.

For example the following types of authentication support this:

http://user:password@www.domain.com ftp://user:password@ftp.domain.com pop3://user:password@mail.domain.com

The port is optional and when not specified libcurl will use the default port based on the determined or specified protocol: 80 for http, 21 for ftp and 25 for smtp, etc. The following examples show how to specify the port:

http://www.weirdserver.com:8080/ - This will connect to a web server using port 8080.

smtp://mail.domain.com:587/ - This will connect to a smtp server on the alternative mail port.

The path part of the URL is protocol specific and whilst some examples are given below this list is not conclusive:

HTTP

The path part of a HTTP request specifies the file to retrieve and from what directory. If the directory is not specified then the web server's root directory is used. If the file is omitted then the default document will be retrieved for either the directory specified or the root directory. The exact resource returned for each URL is entirely dependent on the server's configuration.

http://www.netscape.com - This gets the main page (index.html in this example) from Netscape's web server.

http://www.netscape.com/index.html - This returns the main page from Netscape by specifying the page to get.

http://www.netscape.com/contactus/ - This returns the default document from the contactus directory.

FTP

The path part of an FTP request specifies the file to retrieve and from what directory. If the file part is omitted then libcurl downloads the directory listing for the directory specified. If the directory is omitted then the directory listing for the root / home directory will be returned.

ftp://cool.haxx.se - This retrieves the directory listing for our FTP server.

ftp://cool.haxx.se/readme.txt - This downloads the file readme.txt from the root directory.

ftp://cool.haxx.se/libcurl/readme.txt - This downloads readme.txt from the libcurl directory.

ftp://user:password@my.example.com/readme.txt - This retrieves the readme.txt file from the user's home directory. When a username and password is specified, everything that is specified in the path part is relative to the user's home directory. To retrieve files from the root directory or a directory underneath the root directory then the absolute path must be specified by prepending an additional forward slash to the beginning of the path.

ftp://user:password@my.example.com//readme.txt - This retrieves the readme.txt from the root directory when logging in as a specified user.

SMTP

The path part of a SMTP request specifies the host name to present during communication with the mail server. If the path is omitted then libcurl will attempt to resolve the local computer's host name. However, this may not return the fully qualified domain name that is required by some mail servers and specifying this path allows you to set an alternative name, such as your machine's fully qualified domain name, which you might have obtained from an external function such as gethostname or getaddrinfo.

smtp://mail.domain.com - This connects to the mail server at domain.com and sends your local computer's host name in the HELO / EHLO command.

smtp://mail.domain.com/client.domain.com - This will send client.domain.com in the HELO / EHLO command to the mail server at domain.com.

POP3

The path part of a POP3 request specifies the mailbox (message) to retrieve. If the mailbox is not specified then a list of waiting messages is returned instead.

pop3://user:password@mail.domain.com - This lists the available messages pop3://user:password@mail.domain.com/1 - This retrieves the first message

SCP

The path part of a SCP request specifies the file to retrieve and from what directory. The file part may not be omitted. The file is taken as an absolute path from the root directory on the server. To specify a path relative to the user's home directory on the server, prepend ~/ to the path portion. If the user name is not embedded in the URL, it can be set with the *CURLOPT_USERPWD* or **CURLOPT_USERNAME** option.

scp://user@example.com/etc/issue - This specifies the file /etc/issue

scp://example.com/~/my-file - This specifies the file my-file in the user's home directory on the server

SFTP

The path part of a SFTP request specifies the file to retrieve and from what directory. If the file part is omitted then libcurl downloads the directory listing for the directory specified. If the path ends in a / then a directory listing is returned instead of a file. If the path is omitted entirely then the directory listing for the root / home directory will be returned. If the user name is not embedded in the URL, it can be set with the *CURLOPT_USERPWD* or **CURLOPT_USERNAME** option.

sftp://user:password@example.com/etc/issue - This specifies the file /etc/issue

sftp://user@example.com/~/my-file - This specifies the file my-file in the user's home directory

sftp://ssh.example.com/~/Documents/ - This requests a directory listing of the Documents directory under the user's home directory

NOTES

Starting with version 7.20.0, the fragment part of the URI will not be sent as part of the path, which was previously the case.

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

CURLOPT_PROTOCOLS can be used to limit what protocols libcurl will use for this transfer, independent of what libcurl has been compiled to support. That may be useful if you accept the URL from an external source and want to limit the accessibility.

CURLOPT_PROTOCOLS

Pass a long that holds a bitmask of CURLPROTO_* defines. If used, this bitmask limits what protocols libcurl may use in the transfer. This allows you to have a libcurl built to support a wide range of protocols but still limit specific transfers to only be allowed to use a subset of them. By default libcurl will accept all protocols it supports. See also *CURLOPT_REDIR_PROTOCOLS*. (Added in 7.19.4)

CURLOPT_REDIR_PROTOCOLS

Pass a long that holds a bitmask of CURLPROTO_* defines. If used, this bitmask limits what protocols libcurl may use in a transfer that it follows to in a redirect when *CURLOPT_FOLLOWLOCATION* is enabled. This allows you to limit specific transfers to only be allowed to use a subset of protocols in redirections. By default libcurl will allow all protocols except for FILE and SCP. This is a difference compared to pre-7.19.4 versions which unconditionally would follow to all protocols supported. (Added in 7.19.4)

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. If not specified, libcurl will default to using port 1080 for proxies. *CURLOPT_PROXYPORT*.

When you tell the library to use a 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_HTTPPROXY-TUNNEL*.

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

Setting the proxy string to "" (an empty string) will explicitly disable the use of a proxy, even if there is an environment variable set for it.

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

Since 7.21.7, the proxy string may be specified with a protocol:// prefix to specify alternative proxy protocols. Use socks4://, socks4a://, socks5:// or socks5h:// (the last one to enable socks5 and asking the proxy to do the resolving, also known as CURLPROXY_SOCKS5_HOSTNAME type) to request the specific SOCKS version to be used. No protocol specified, http:// and all others will be treated as HTTP proxies.

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_HTTP_1_0* (added in 7.19.4), *CURLPROXY_SOCKS4* (added in 7.10), *CURLPROXY_SOCKS5*, *CURLPROXY_SOCKS4A* (added in 7.18.0) and *CURLPROXY_SOCKS5_HOSTNAME* (added in 7.18.0). The HTTP type is default. (Added in 7.10)

If you set **CURLOPT_PROXYTYPE** to *CURLPROXY_HTTP_1_0*, it will only affect how libcurl speaks to a proxy when CONNECT is used. The HTTP version used for "regular" HTTP requests is instead controlled with *CURLOPT_HTTP_VERSION*.

CURLOPT_NOPROXY

Pass a pointer to a zero terminated string. This should be a comma separated list of hosts which do not use a proxy, if one is specified. The only wildcard is a single * character, which matches all hosts, and effectively disables the proxy. Each name in this list is matched as either a domain which contains the hostname, or the hostname itself. For example, local.com would match local.com, local.com:80, and www.local.com, but not www.notlocal.com. (Added in 7.19.4)

CURLOPT_HTTPPROXYTUNNEL

Set the parameter to 1 to make the library 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_SOCKS5_GSSAPI_SERVICE

Pass a char * as parameter to a string holding the name of the service. The default service name for a SOCKS5 server is rcmd/server-fqdn. This option allows you to change it. (Added in 7.19.4)

CURLOPT_SOCKS5_GSSAPI_NEC

Pass a long set to 1 to enable or 0 to disable. As part of the gssapi negotiation a protection mode is negotiated. The rfc1961 says in section 4.3/4.4 it should be protected, but the NEC reference implementation does not. If enabled, this option allows the unprotected exchange of the protection mode negotiation. (Added in 7.19.4).

CURLOPT_INTERFACE

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

Starting with 7.24.0: If the parameter starts with "if!" then it is treated as only as interface name and no attempt will ever be named to do treat it as an IP address or to do name resolution on it. If the parameter starts with "host!" it is treated as either an IP address or a hostname. Hostnames are resolved synchronously. Using the if! format is highly recommended when using the multi interfaces to avoid allowing the code to block. If "if!" is specified but the parameter does not match an existing interface, CURLE_INTERFACE_FAILED is returned.

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. Valid port numbers are 1 - 65535. (Added in 7.15.2)

CURLOPT_LOCALPORTRANGE

Pass a long. This is the number of attempts libcurl should make 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 to 1 or below will make libcurl do only one try for the exact port number. Port numbers by nature are scarce resources 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 to completely disable caching, or set to -1 to make the cached entries remain forever. By default, libcurl caches this info for 60 seconds.

The name resolve functions of various libc implementations don't re-read name server information unless explicitly told so (for example, by calling *res_init(3)*). This may cause libcurl to keep using the older server even if DHCP has updated the server info, and this may look like a DNS cache issue to the casual libcurl-app user.

CURLOPT_DNS_USE_GLOBAL_CACHE

Pass a long. If the value is 1, 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 makes 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.

CURLOPT_ADDRESS_SCOPE

Pass a long specifying the scope_id value to use when connecting to IPv6 link-local or site-local addresses. (Added in 7.19.0)

CURLOPT_TCP_KEEPALIVE

Pass a long. If set to 1, TCP keepalive probes will be sent. The delay and frequency of these probes can be controlled by the *CURLOPT_TCP_KEEPIDLE* and *CURLOPT_TCP_KEEPINTVL* options, provided the operating system supports them. Set to 0 (default behavior) to disable keepalive probes (Added in 7.25.0).

CURLOPT_TCP_KEEPIDLE

Pass a long. Sets the delay, in seconds, that the operating system will wait while the connection is idle before sending keepalive probes. Not all operating systems support this option. (Added in 7.25.0)

CURLOPT_TCP_KEEPINTVL

Pass a long. Sets the interval, in seconds, that the operating system will wait between sending keepalive probes. Not all operating systems support this option. (Added in 7.25.0)

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 for the host and user name (to find the password only) or for 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 for 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 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 the authentication method.

When using NTLM, you can set the 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 the authentication method.

CURLOPT_USERNAME

Pass a char * as parameter, which should be pointing to the zero terminated user name to use for the transfer.

CURLOPT_USERNAME sets the user name to be used in protocol authentication. You should not use this option together with the (older) CURLOPT_USERPWD option.

In order to specify the password to be used in conjunction with the user name use the *CURLOPT_PASSWORD* option. (Added in 7.19.1)

CURLOPT_PASSWORD

Pass a char * as parameter, which should be pointing to the zero terminated password to use for the transfer.

The CURLOPT_PASSWORD option should be used in conjunction with the *CURLOPT_USERNAME* option. (Added in 7.19.1)

CURLOPT_PROXYUSERNAME

Pass a char * as parameter, which should be pointing to the zero terminated user name to use for the transfer while connecting to Proxy.

The CURLOPT_PROXYUSERNAME option should be used in same way as the *CURLOPT_PROXYUSERPWD* is used. In comparison to *CURLOPT_PROXYUSERPWD* the CURLOPT_PROXYUSERNAME allows the username to contain a colon, like in the following example: "sip:user@example.com". The CURLOPT_PROXYUSERNAME option is an alternative way to set the user name while connecting to Proxy. There is no meaning to use it together with the *CURLOPT_PROXYUSERPWD* option.

In order to specify the password to be used in conjunction with the user name use the *CURLOPT_PROXYPASSWORD* option. (Added in 7.19.1)

CURLOPT_PROXYPASSWORD

Pass a char * as parameter, which should be pointing to the zero terminated password to use for the transfer while connecting to Proxy.

The CURLOPT_PROXYPASSWORD option should be used in conjunction with the *CURLOPT_PROXYUSERNAME* option. (Added in 7.19.1)

CURLOPT_HTTPAUTH

Pass a long as parameter, which is set to a bitmask, to tell libcurl which 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 which 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 or with the *CURLOPT_USERNAME* and the *CURLOPT_PASSWORD* options. (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 sends 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_DIGEST_IE

HTTP Digest authentication with an IE flavor. 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. The IE flavor is simply that libcurl will use a special "quirk" that IE is known to have used before version 7 and that some servers require the client to use. (This define was added in 7.19.3)

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 also be used along with other 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 either OpenSSL, GnuTLS or NSS support for this option to work, or build libcurl on Windows.

CURLAUTH_NTLM_WB

NTLM delegating to winbind helper. Authentication is performed by a separate binary application that is executed when needed. The name of the application is specified at compile time but is typically /usr/bin/ntlm_auth (Added in 7.22.0)

Note that libcurl will fork when necessary to run the winbind application and kill it when complete, calling waitpid() to await its exit when done. On POSIX operating systems,

killing the process will cause a SIGCHLD signal to be raised (regardless of whether *CURLOPT_NOSIGNAL* is set), which must be handled intelligently by the application. In particular, the application must not unconditionally call wait() in its SIGCHLD signal handler to avoid being subject to a race condition. This behavior is subject to change in future versions of libcurl.

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.

CURLAUTH_ONLY
: This is a meta symbol. Or this value together with a single specific auth value to force libcurl to probe for un-restricted auth and if not, only that single auth algorithm is acceptable. (Added in 7.21.3)

CURLOPT_TLSAUTH_TYPE
: Pass a long as parameter, which is set to a bitmask, to tell libcurl which authentication method(s) you want it to use for TLS authentication.

CURLOPT_TLSAUTH_SRP
: TLS-SRP authentication. Secure Remote Password authentication for TLS is defined in RFC 5054 and provides mutual authentication if both sides have a shared secret. To use TLS-SRP, you must also set the *CURLOPT_TLSAUTH_USERNAME* and *CURLOPT_TLSAUTH_PASSWORD* options.

  You need to build libcurl with GnuTLS or OpenSSL with TLS-SRP support for this to work. (Added in 7.21.4)

CURLOPT_TLSAUTH_USERNAME
: Pass a char * as parameter, which should point to the zero terminated username to use for the TLS authentication method specified with the *CURLOPT_TLSAUTH_TYPE* option. Requires that the *CURLOPT_TLS_PASSWORD* option also be set. (Added in 7.21.4)

CURLOPT_TLSAUTH_PASSWORD
: Pass a char * as parameter, which should point to the zero terminated password to use for the TLS authentication method specified with the *CURLOPT_TLSAUTH_TYPE* option. Requires that the *CURLOPT_TLS_USERNAME* option also be set. (Added in 7.21.4)

CURLOPT_PROXYAUTH
: Pass a long as parameter, which is set to a bitmask, to tell libcurl which 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 parameter set to 1 to enable this. When enabled, libcurl will automatically set the Referer: field in requests where it follows a Location: redirect.

CURLOPT_ACCEPT_ENCODING
: Sets the contents of the Accept-Encoding: header sent in a 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.

(This option was called CURLOPT_ENCODING before 7.21.6)

CURLOPT_TRANSFER_ENCODING

Adds a request for compressed Transfer Encoding in the outgoing HTTP request. If the server supports this and so desires, it can respond with the HTTP resonse sent using a compressed Transfer-Encoding that will be automatically uncompressed by libcurl on receival.

Transfer-Encoding differs slightly from the Content-Encoding you ask for with **CURLOPT_ACCEPT_ENCODING** in that a Transfer-Encoding is strictly meant to be for the transfer and thus MUST be decoded before the data arrives in the client. Traditionally, Transfer-Encoding has been much less used and supported by both HTTP clients and HTTP servers.

(Added in 7.21.6)

CURLOPT_FOLLOWLOCATION

A parameter set to 1 tells the library to follow any Location: header that the server sends as part of a 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.

Since 7.19.4, libcurl can limit what protocols it will automatically follow. The accepted protocols are set with *CURLOPT_REDIR_PROTOCOLS* and it excludes the FILE protocol by default.

CURLOPT_UNRESTRICTED_AUTH

A parameter set to 1 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_POSTREDIR

Pass a bitmask to control how libcurl acts on redirects after POSTs that get a 301, 302 or 303 response back. A parameter with bit 0 set (value **CURL_REDIR_POST_301**) tells the library to respect RFC 2616/10.3.2 and not convert POST requests into GET requests when following a 301 redirection. Setting bit 1 (value CURL_REDIR_POST_302) makes libcurl maintain the request method after a 302 redirect. Setting bit 2 (value **CURL_REDIR_POST_303) makes libcurl maintain the request method after a 302 redirect. CURL_REDIR_POST_ALL is a convenience define that sets both bits.**

The non-RFC behaviour is ubiquitous in web browsers, so the library does the conversion by default to maintain consistency. However, a server may require a POST to remain a POST after such a redirection. This option is meaningful only when setting *CURLOPT_FOLLOWLOCATION***. (Added in 7.17.1) (This option was known as CURLOPT_POST301 up to**

7.19.0 as it only supported the 301 way before then)

CURLOPT_PUT

A parameter set to 1 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 parameter set to 1 tells the library to do a regular HTTP post. This will also make the library use a "Content-Type: application/x-www-form-urlencoded" header. (This is by far the most commonly used POST method).

Use one of *CURLOPT_POSTFIELDS* or *CURLOPT_COPYPOSTFIELDS* options to specify what data to post and *CURLOPT_POSTFIELDSIZE* or *CURLOPT_POSTFIELDSIZE_LARGE* 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* or *CURLOPT_POSTFIELDSIZE_LARGE* option. To enable chunked encoding, you simply pass in the appropriate Transfer-Encoding header, see the post-callback.c example.

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 1, 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 explicitly set the new request type using *CURLOPT_NOBODY* or *CURLOPT_HTTPGET* or similar.

CURLOPT_POSTFIELDS

Pass a void * as parameter, which should be the full data to post in a 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.

The pointed data are NOT copied by the library: as a consequence, they must be preserved by the calling application until the transfer finishes.

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*.

If you want to do a zero-byte POST, you need to set *CURLOPT_POSTFIELDSIZE* explicitly to zero, as simply setting *CURLOPT_POSTFIELDS* to NULL or "" just effectively disables the sending of the specified string. libcurl will instead assume that you'll send the POST data using the read callback!

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 RFC2388-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_COPYPOSTFIELDS

Pass a char * as parameter, which should be the full data to post in a HTTP POST operation. It behaves as the *CURLOPT_POSTFIELDS* option, but the original data are copied by the library, allowing the application to overwrite the original data after setting this option.

Because data are copied, care must be taken when using this option in conjunction with *CURLOPT_POSTFIELDSIZE* or *CURLOPT_POSTFIELDSIZE_LARGE*: If the size has not been set prior to *CURLOPT_COPYPOSTFIELDS*, the data are assumed to be a NUL-terminated string; else the stored size informs the library about the data byte count to copy. In any case, the size must not be changed after *CURLOPT_COPYPOSTFIELDS*, unless another *CURLOPT_POSTFIELDS* or *CURLOPT_COPYPOSTFIELDS* option is issued. (Added in 7.17.1)

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 content 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 content, make the content 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. Before libcurl 7.16.3, Libcurl used the value set by option *CURLOPT_HTTP_VERSION*, but starting with 7.16.3 the protocol is assumed to match HTTP 1.0 when an alias matched.

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.

This option sets the cookie header explicitly in the outgoing request(s). If multiple requests are done due to authentication, followed redirections or similar, they will all get this cookie passed on.

Using this option multiple times will only make the latest string override the previous 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 requests.

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 1 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 or 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) Passing the special string "FLUSH" will write all cookies known by cURL to the file specified by *CURLOPT_COOKIEJAR*. (Added in 7.17.1)

CURLOPT_HTTPGET

Pass a long. If the long is 1, this forces the HTTP request to get back to GET. Usable if a POST, HEAD, PUT, or a custom request has been used previously using the same curl handle.

When setting *CURLOPT_HTTPGET* to 1, 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)

CURLOPT_HTTP_CONTENT_DECODING

Pass a long to tell libcurl how to act on content decoding. If set to zero, content decoding will be disabled. If set to 1 it is enabled. Libcurl has no default content decoding but requires you to use *CURLOPT_ENCODING* for that. (added in 7.16.2)

CURLOPT_HTTP_TRANSFER_DECODING

Pass a long to tell libcurl how to act on transfer decoding. If set to zero, transfer decoding will be disabled, if set to 1 it is enabled (default). libcurl does chunked transfer decoding by default unless

this option is set to zero. (added in 7.16.2)

SMTP OPTIONS

CURLOPT_MAIL_FROM

Pass a pointer to a zero terminated string as parameter. This should be used to specify the sender's email address when sending SMTP mail with libcurl.

An originator email address should be specified with angled brackets (<>) around it, which if not specified, will be added by libcurl from version 7.21.4 onwards. Failing to provide such brackets may cause the server to reject the email.

If this parameter is not specified then an empty address will be sent to the mail server which may or may not cause the email to be rejected.

(Added in 7.20.0)

CURLOPT_MAIL_RCPT

Pass a pointer to a linked list of recipients to pass to the server in your SMTP mail 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.

Each recipient should be specified within a pair of angled brackets (<>), however, should you not use an angled bracket as the first character libcurl will assume you provided a single email address and enclose that address within brackets for you.

(Added in 7.20.0)

CURLOPT_MAIL_AUTH

Pass a pointer to a zero terminated string as parameter. This will be used to specify the authentication address (identity) of a submitted message that is being relayed to another server.

This optional parameter allows co-operating agents in a trusted environment to communicate the authentication of individual messages and should only be used by the application program, using libcurl, if the application is itself a mail server acting in such an environment. If the application is operating as such and the AUTH address is not known or is invalid, then an empty string should be used for this parameter.

Unlike CURLOPT_MAIL_FROM and CURLOPT_MAIL_RCPT, the address should not be specified within a pair of angled brackets (<>). However, if an empty string is used then a pair of brackets will be sent by libcurl as required by RFC-2554.

(Added in 7.24.0)

TFTP OPTIONS

CURLOPT_TFTP_BLKSIZE

Specify block size to use for TFTP data transmission. Valid range as per RFC 2348 is 8-65464 bytes. The default of 512 bytes will be used if this option is not specified. The specified block size will only be used pending support by the remote server. If the server does not return an option acknowledgement or returns an option acknowledgement with no blksize, the default of 512 bytes will be used. (added in 7.19.4)

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, a network interface name (under Unix) or just a '-' symbol to let the library use your system's default IP address. Default

FTP operations are passive, and thus won't use PORT.

The address can be followed by a ':' to specify a port, optionally followed by a '-' to specify a port range. If the port specified is 0, the operating system will pick a free port. If a range is provided and all ports in the range are not available, libcurl will report CURLE_FTP_PORT_FAILED for the handle. Invalid port/range settings are ignored. IPv6 addresses followed by a port or portrange have to be in brackets. IPv6 addresses without port/range specifier can be in brackets. (added in 7.19.5)

Examples with specified ports:

```
eth0:0
192.168.1.2:32000-33000
curl.se:32123
[::1]:1234-4567
```

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 or SFTP commands to pass to the server prior to your FTP request. This will be done before any other commands are issued (even before the CWD command for FTP). The linked list should be a fully valid list of 'struct curl_slist' structs properly filled in with text strings. Use *curl_slist_append(3)* 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. When speaking to a FTP (or SFTP since 7.24.0) server, prefix the command with an asterisk (*) to make libcurl continue even if the command fails as by default libcurl will stop at first failure.

The set of valid FTP commands depends on the server (see RFC959 for a list of mandatory commands).

The valid SFTP commands are: chgrp, chmod, chown, ln, mkdir, pwd, rename, rm, rmdir, symlink (see **curl**(1)) (SFTP support added in 7.16.3)

CURLOPT_POSTQUOTE

Pass a pointer to a linked list of FTP or SFTP commands to pass to the server after your FTP transfer request. The commands will only be run if no error occurred. 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. Before version 7.16.0, if you also set *CURLOPT_NOBODY* to 1, this option didn't work.

CURLOPT_DIRLISTONLY

A parameter set to 1 tells the library to just list the names of files in a directory, instead of doing a full directory listing that would include file sizes, dates etc. This works for FTP and SFTP URLs.

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

Setting this option to 1 also implies a directory listing even if the URL doesn't end with a slash, which otherwise is necessary.

Do NOT use this option if you also use *CURLOPT_WILDCARDMATCH* as it will effectively

break that feature then.

(This option was known as CURLOPT_FTPLISTONLY up to 7.16.4)

CURLOPT_APPEND
A parameter set to 1 tells the library to append to the remote file instead of overwrite it. This is only useful when uploading to an FTP site.

(This option was known as CURLOPT_FTPAPPEND up to 7.16.4)

CURLOPT_FTP_USE_EPRT
Pass a long. If the value is 1, 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 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 1, 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 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_USE_PRET
Pass a long. If the value is 1, it tells curl to send a PRET command before PASV (and EPSV). Certain FTP servers, mainly drftpd, require this non-standard command for directory listings as well as up and downloads in PASV mode. Has no effect when using the active FTP transfers mode. (Added in 7.20.0)

CURLOPT_FTP_CREATE_MISSING_DIRS
Pass a long. If the value is 1, 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)

This setting also applies to SFTP-connections. curl will attempt to create the remote directory if it can't obtain a handle to the target-location. The creation will fail if a file of the same name as the directory to create already exists or lack of permissions prevents creation. (Added in 7.16.3)

Starting with 7.19.4, you can also set this value to 2, which will make libcurl retry the CWD command again if the subsequent MKD command fails. This is especially useful if you're doing many simultanoes connections against the same server and they all have this option enabled, as then CWD may first fail but then another connection does MKD before this connection and thus MKD fails but trying CWD works! 7.19.4 also introduced the *CURLFTP_CREATE_DIR* and *CURLFTP_CREATE_DIR_RETRY* enum names for these arguments.

Before version 7.19.4, libcurl will simply ignore arguments set to 2 and act as if 1 was selected.

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_ALTERNATIVE_TO_USER
: Pass a char * as parameter, pointing to a string which will be used to authenticate if the usual FTP "USER user" and "PASS password" negotiation fails. This is currently only known to be required when connecting to Tumbleweed's Secure Transport FTPS server using client certificates for authentication. (Added in 7.15.5)

CURLOPT_FTP_SKIP_PASV_IP
: Pass a long. If set to 1, 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_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_USE_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_FTP_SSL_CCC
: If enabled, this option makes libcurl use CCC (Clear Command Channel). It shuts down the SSL/TLS layer after authenticating. The rest of the control channel communication will be unencrypted. This allows NAT routers to follow the FTP transaction. Pass a long using one of the values below. (Added in 7.16.1)

  CURLFTPSSL_CCC_NONE
  : Don't attempt to use CCC.

  CURLFTPSSL_CCC_PASSIVE
  : Do not initiate the shutdown, but wait for the server to do it. Do not send a reply.

  CURLFTPSSL_CCC_ACTIVE
  : Initiate the shutdown and wait for a reply.

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 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'.

(Added in 7.15.1)

RTSP OPTIONS

CURLOPT_RTSP_REQUEST

Tell libcurl what kind of RTSP request to make. Pass one of the following RTSP enum values. Unless noted otherwise, commands require the Session ID to be initialized. (Added in 7.20.0)

CURL_RTSPREQ_OPTIONS

Used to retrieve the available methods of the server. The application is responsbile for parsing and obeying the response. **(The session ID is not needed for this method.)** (Added in 7.20.0)

CURL_RTSPREQ_DESCRIBE

Used to get the low level description of a stream. The application should note what formats it understands in the *'Accept:'* header. Unless set manually, libcurl will automatically fill in *'Accept: application/sdp'*. Time-condition headers will be added to Describe requests if the *CURLOPT_TIMECONDITION* option is active. **(The session ID is not needed for this method)** (Added in 7.20.0)

CURL_RTSPREQ_ANNOUNCE

When sent by a client, this method changes the description of the session. For example, if a client is using the server to record a meeting, the client can use Announce to inform the server of all the meta-information about the session. ANNOUNCE acts like a HTTP PUT or POST just like *CURL_RTSPREQ_SET_PARAMETER* (Added in 7.20.0)

CURL_RTSPREQ_SETUP

Setup is used to initialize the transport layer for the session. The application must set the desired Transport options for a session by using the *CURLOPT_RTSP_TRANSPORT* option prior to calling setup. If no session ID is currently set with *CURLOPT_RTSP_SESSION_ID*, libcurl will extract and use the session ID in the response to this request. **(The session ID is not needed for this method).** (Added in 7.20.0)

CURL_RTSPREQ_PLAY

Send a Play command to the server. Use the *CURLOPT_RANGE* option to modify the playback time (e.g. 'npt=10-15'). (Added in 7.20.0)

CURL_RTSPREQ_PAUSE

Send a Pause command to the server. Use the *CURLOPT_RANGE* option with a single value to indicate when the stream should be halted. (e.g. npt='25') (Added in 7.20.0)

CURL_RTSPREQ_TEARDOWN

This command terminates an RTSP session. Simply closing a connection does not terminate the RTSP session since it is valid to control an RTSP session over different connections. (Added in 7.20.0)

CURL_RTSPREQ_GET_PARAMETER

Retrieve a parameter from the server. By default, libcurl will automatically include a *Content-Type: text/parameters* header on all non-empty requests unless a custom one is set. GET_PARAMETER acts just like a HTTP PUT or POST (see *CURL_RTSPREQ_SET_PARAMETER*). Applications wishing to send a heartbeat message (e.g. in the presence of a server-specified timeout) should send use an empty GET_PARAMETER request. (Added in 7.20.0)

CURL_RTSPREQ_SET_PARAMETER

Set a parameter on the server. By default, libcurl will automatically include a *Content-Type: text/parameters* header unless a custom one is set. The interaction with

SET_PARAMTER is much like a HTTP PUT or POST. An application may either use *CURLOPT_UPLOAD* with *CURLOPT_READDATA* like a HTTP PUT, or it may use *CURLOPT_POSTFIELDS* like a HTTP POST. No chunked transfers are allowed, so the application must set the *CURLOPT_INFILESIZE* in the former and *CURLOPT_POSTFIELDSIZE* in the latter. Also, there is no use of multi-part POSTs within RTSP. (Added in 7.20.0)

CURL_RTSPREQ_RECORD
Used to tell the server to record a session. Use the *CURLOPT_RANGE* option to modify the record time. (Added in 7.20.0)

CURL_RTSPREQ_RECEIVE
This is a special request because it does not send any data to the server. The application may call this function in order to receive interleaved RTP data. It will return after processing one read buffer of data in order to give the application a chance to run. (Added in 7.20.0)

CURLOPT_RTSP_SESSION_ID
Pass a char * as a parameter to set the value of the current RTSP Session ID for the handle. Useful for resuming an in-progress session. Once this value is set to any non-NULL value, libcurl will return *CURLE_RTSP_SESSION_ERROR* if ID received from the server does not match. If unset (or set to NULL), libcurl will automatically set the ID the first time the server sets it in a response. (Added in 7.20.0)

CURLOPT_RTSP_STREAM_URI
Set the stream URI to operate on by passing a char * . For example, a single session may be controlling *rtsp://foo/twister/audio* and *rtsp://foo/twister/video* and the application can switch to the appropriate stream using this option. If unset, libcurl will default to operating on generic server options by passing '*' in the place of the RTSP Stream URI. This option is distinct from *CURLOPT_URL*. When working with RTSP, the *CURLOPT_STREAM_URI* indicates what URL to send to the server in the request header while the *CURLOPT_URL* indicates where to make the connection to. (e.g. the *CURLOPT_URL* for the above examples might be set to *rtsp://foo/twister* (Added in 7.20.0)

CURLOPT_RTSP_TRANSPORT
Pass a char * to tell libcurl what to pass for the Transport: header for this RTSP session. This is mainly a convenience method to avoid needing to set a custom Transport: header for every SETUP request. The application must set a Transport: header before issuing a SETUP request. (Added in 7.20.0)

CURLOPT_RTSP_HEADER
This option is simply an alias for *CURLOPT_HTTP_HEADER*. Use this to replace the standard headers that RTSP and HTTP share. It is also valid to use the shortcuts such as *CURLOPT_USERAGENT*. (Added in 7.20.0)

CURLOPT_RTSP_CLIENT_CSEQ
Manually set the the CSEQ number to issue for the next RTSP request. Useful if the application is resuming a previously broken connection. The CSEQ will increment from this new number henceforth. (Added in 7.20.0)

CURLOPT_RTSP_SERVER_CSEQ
Manually set the CSEQ number to expect for the next RTSP Server->Client request. At the moment, this feature (listening for Server requests) is unimplemented. (Added in 7.20.0)

PROTOCOL OPTIONS

CURLOPT_TRANSFERTEXT
A parameter set to 1 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_PROXY_TRANSFER_MODE

Pass a long. If the value is set to 1 (one), it tells libcurl to set the transfer mode (binary or ASCII) for FTP transfers done via a HTTP proxy, by appending ;type=a or ;type=i to the URL. Without this setting, or it being set to 0 (zero, the default), *CURLOPT_TRANSFERTEXT* has no effect when doing FTP via a proxy. Beware that not all proxies support this feature. (Added in 7.18.0)

CURLOPT_CRLF

Pass a long. If the value is set to 1 (one), libcurl converts Unix newlines to CRLF newlines on transfers. Disable this option again by setting the value to 0 (zero).

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). For RTSP, the formatting of a range should follow RFC 2326 Section 12.29. For RTSP, byte ranges are **not** permitted. Instead, ranges should be given in npt, utc, or smpte formats.

Pass a NULL to this option to disable the use of ranges.

Ranges work on HTTP, FTP, FILE (since 7.18.0), and RTSP (since 7.20.0) transfers only.

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). For FTP, set this option to -1 to make the transfer start from the end of the target file (useful to continue an interrupted upload).

When doing uploads with FTP, the resume position is where in the local/source file libcurl should try to resume the upload from and it will then append the source file to the remote target file.

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 can be used to specify the request instead of GET or HEAD when performing HTTP based requests, instead of LIST and NLST when performing FTP directory listings and instead of LIST and RETR when issuing POP3 based commands. This is particularly useful, for example, for performing a HTTP DELETE request or a POP3 DELE command.

Please don’t perform this at will, on HTTP based requests, by making sure your server supports the command you are sending first.

When you change the request method by setting **CURLOPT_CUSTOMREQUEST** to something, you don’t actually change how libcurl behaves or acts in regards to the particular request method, it will only change the actual string sent in the request.

For example: if you tell libcurl to do a HEAD request, but then change the request to a "GET" with **CURLOPT_CUSTOMREQUEST** you’ll still see libcurl act as if it sent a HEAD even when it does send a GET.

To switch to a proper HEAD, use *CURLOPT_NOBODY*, to switch to a proper POST, use

CURLOPT_POST or *CURLOPT_POSTFIELDS* and so on.

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.

(Support for POP3 added in 7.26.0)

CURLOPT_FILETIME

Pass a long. If it is 1, 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 parameter set to 1 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*.

For uploading using SCP, this option or *CURLOPT_INFILESIZE_LARGE* is mandatory.

When sending emails using SMTP, this command can be used to specify the optional SIZE parameter for the MAIL FROM command. (Added in 7.23.0)

This option does not limit how much data libcurl will actually send, as that is controlled entirely by what the read callback returns.

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)

For uploading using SCP, this option or *CURLOPT_INFILESIZE* is mandatory.

This option does not limit how much data libcurl will actually send, as that is controlled entirely by what the read callback returns.

CURLOPT_UPLOAD

A parameter set to 1 tells the library to prepare for an upload. The *CURLOPT_READDATA* and *CURLOPT_INFILESIZE* or *CURLOPT_INFILESIZE_LARGE* options 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, FTP, RTSP, and FILE.

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 not have been met. *curl_easy_getinfo(3)* with the *CURLINFO_CONDITION_UNMET* option can be used after a transfer to learn if a zero-byte successful "transfer" was due to this condition not matching.

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.

Default timeout is 0 (zero) which means it never times out.

CURLOPT_TIMEOUT_MS

Like *CURLOPT_TIMEOUT* but takes number of milliseconds instead. If libcurl is built to use the standard system name resolver, that portion of the transfer will still use full-second resolution for timeouts with a minimum timeout allowed of one second. (Added in 7.16.2)

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_MAX_SEND_SPEED_LARGE
: Pass a curl_off_t as parameter. If an upload exceeds this speed (counted in bytes per second) on cumulative average during the transfer, the transfer will pause to keep the average rate less than or equal to the parameter value. Defaults to unlimited speed. (Added in 7.15.5)

CURLOPT_MAX_RECV_SPEED_LARGE
: Pass a curl_off_t as parameter. If a download exceeds this speed (counted in bytes per second) on cumulative average during the transfer, the transfer will pause to keep the average rate less than or equal to the parameter value. Defaults to unlimited speed. (Added in 7.15.5)

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 in this easy handle. Default is 5, and there isn't much point in changing this value unless you are perfectly aware of how this works and changes libcurl's behaviour. This concerns connections using any of the protocols that support persistent connections.

  When reaching the maximum limit, curl closes the oldest one in the cache to prevent increasing the number of open connections.

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

  If you add this easy handle to a multi handle, this setting is not acknowledged, and you must instead use *curl_multi_setopt(3)* and the *CURLMOPT_MAXCONNECTS* option.

CURLOPT_CLOSEPOLICY
: (Obsolete) This option does nothing.

CURLOPT_FRESH_CONNECT
: Pass a long. Set to 1 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 1 to make the next transfer explicitly close the connection when done. Normally, libcurl keeps all connections alive when done with one transfer in case a succeeding one follows 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 possible 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 switch to the default built-in connection timeout - 300 seconds. See also the *CURLOPT_TIMEOUT* option.

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

CURLOPT_CONNECTTIMEOUT_MS
: Like *CURLOPT_CONNECTTIMEOUT* but takes the number of milliseconds instead. If libcurl is built to use the standard system name resolver, that portion of the connect will still use full-second resolution for timeouts with a minimum timeout allowed of one second. (Added in 7.16.2)

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. If the parameter equals 1, it tells the library to perform all the required proxy authentication and connection setup, but no data transfer. This option is useful only on HTTP URLs.

  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)

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

  This is for enabling SSL/TLS when you use FTP, SMTP, POP3, IMAP etc.

  (This option was known as CURLOPT_FTP_SSL up to 7.16.4, and the constants were known as CURLFTPSSL_*)

  CURLUSESSL_NONE
  : Don’t attempt to use SSL.

  CURLUSESSL_TRY
  : Try using SSL, proceed as normal otherwise.

  CURLUSESSL_CONTROL
  : Require SSL for the control connection or fail with *CURLE_USE_SSL_FAILED*.

  CURLUSESSL_ALL
  : Require SSL for all communication or fail with *CURLE_USE_SSL_FAILED*.

CURLOPT_RESOLVE
: Pass a pointer to a linked list of strings with host name resolve information to use for requests with this handle. 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.

  Each single name resolve string should be written using the format HOST:PORT:ADDRESS where HOST is the name libcurl will try to resolve, PORT is the port number of the service where libcurl wants to connect to the HOST and ADDRESS is the numerical IP address. If libcurl is built to support IPv6, ADDRESS can of course be either IPv4 or IPv6 style addressing.

  This option effectively pre-populates the DNS cache with entries for the host+port pair so redirects and everything that operations against the HOST+PORT will instead use your provided ADDRESS.

  You can remove names from the DNS cache again, to stop providing these fake resolves, by including a string in the linked list that uses the format "-HOST:PORT". The host name must be prefixed with a dash, and the host name and port number must exactly match what was already

added previously.

(Added in 7.21.3)

CURLOPT_DNS_SERVERS
Set the list of DNS servers to be used instead of the system default. The format of the dns servers option is:

host[:port][,host[:port]]...

For example:

192.168.1.100,192.168.1.101,3.4.5.6

This option requires that libcurl was built with a resolver backend that supports this operation. The c-ares backend is the only such one.

(Added in 7.24.0)

CURLOPT_ACCEPTTIMEOUT_MS
Pass a long telling libcurl the maximum number of milliseconds to wait for a server to connect back to libcurl when an active FTP connection is used. If no timeout is set, the internal default of 60000 will be used. (Added in 7.24.0)

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*.

With NSS this can also be the nickname of the certificate you wish to authenticate with. If you want to use a file from the current directory, please precede it with "./" prefix, in order to avoid confusion with a nickname.

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_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_KEYPASSWD
Pass a pointer to a zero terminated string as parameter. It will be used as the password required to use the *CURLOPT_SSLKEY* or *CURLOPT_SSH_PRIVATE_KEYFILE* private key. You never needed a pass phrase to load a certificate but you need one to load your private key.

(This option was known as CURLOPT_SSLKEYPASSWD up to 7.16.4 and CURLOPT_SSLCERTPASSWD up to 7.9.2)

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.

Even though this option doesn't need any parameter, in some configurations *curl_easy_setopt* might be defined as a macro taking exactly three arguments. Therefore, it's recommended to pass 1 as parameter to this option.

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. This will attempt to figure out the remote SSL protocol version, i.e. either SSLv3 or TLSv1 (but not SSLv2, which became disabled by default with 7.18.1).

CURL_SSLVERSION_TLSv1

Force TLSv1

CURL_SSLVERSION_SSLv2

Force SSLv2

CURL_SSLVERSION_SSLv3

Force SSLv3

CURLOPT_SSL_VERIFYPEER

Pass a long as parameter. By default, curl assumes a value of 1.

This option determines whether curl verifies the authenticity of the peer's certificate. A value of 1 means curl verifies; 0 (zero) means it doesn't.

When negotiating a 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. curl uses a default bundle of CA certificates (the path for that is determined at build time) 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 peer certificate verification 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. The check that the host name in the certificate is valid for the host name you're connecting to is done independently of the *CURLOPT_SSL_VERIFYPEER* option.

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.

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

When built against NSS, this is the directory that the NSS certificate database resides in.

CURLOPT_ISSUERCERT

Pass a char * to a zero terminated string naming a file holding a CA certificate in PEM format. If the option is set, an additional check against the peer certificate is performed to verify the issuer is indeed the one associated with the certificate provided by the option. This additional check is useful in multi-level PKI where one needs to enforce that the peer certificate is from a specific branch of the tree.

This option makes sense only when used in combination with the *CURLOPT_SSL_VERIFYPEER* option. Otherwise, the result of the check is not considered as failure.

A specific error code (CURLE_SSL_ISSUER_ERROR) is defined with the option, which is returned if the setup of the SSL/TLS session has failed due to a mismatch with the issuer of peer certificate (*CURLOPT_SSL_VERIFYPEER* has to be set too for the check to fail). (Added in 7.19.0)

CURLOPT_CAPATH

Pass a char * to a zero terminated string naming a directory holding multiple CA certificates to verify the peer with. If libcurl is built against OpenSSL, 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. This option is OpenSSL-specific and does nothing if libcurl is built to use GnuTLS. NSS-powered libcurl provides the option only for backward compatibility.

CURLOPT_CRLFILE

Pass a char * to a zero terminated string naming a file with the concatenation of CRL (in PEM format) to use in the certificate validation that occurs during the SSL exchange.

When curl is built to use NSS or GnuTLS, there is no way to influence the use of CRL passed to help in the verification process. When libcurl is built with OpenSSL support, X509_V_FLAG_CRL_CHECK and X509_V_FLAG_CRL_CHECK_ALL are both set, requiring CRL check against all the elements of the certificate chain if a CRL file is passed.

This option makes sense only when used in combination with the *CURLOPT_SSL_VERIFYPEER* option.

A specific error code (CURLE_SSL_CRL_BADFILE) is defined with the option. It is returned when the SSL exchange fails because the CRL file cannot be loaded. A failure in certificate verification due to a revocation information found in the CRL does not trigger this specific error. (Added in 7.19.0)

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 a 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 value for this option is 2.

This option controls checking the server's certificate's claimed identity. The server could be lying. To control lying, see *CURLOPT_SSL_VERIFYPEER*. If libcurl is built against NSS and *CURLOPT_SSL_VERIFYPEER* is zero, *CURLOPT_SSL_VERIFYHOST* is ignored.

CURLOPT_CERTINFO

Pass a long set to 1 to enable libcurl's certificate chain info gatherer. With this enabled, libcurl (if built with OpenSSL) will extract lots of information and data about the certificates in the certificate chain used in the SSL connection. This data is then possible to extract after a transfer using *curl_easy_getinfo(3)* and its option *CURLINFO_CERTINFO*. (Added in 7.19.1)

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_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.

For OpenSSL and GnuTLS 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*

For NSS, valid examples of cipher lists include 'rsa_rc4_128_md5', ´rsa_aes_128_sha´, etc. With NSS you don't add/remove ciphers. If one uses this option then all known ciphers are disabled and only those passed in are enabled.

You'll find more details about the NSS cipher lists on this URL: *http://directory.fedora.redhat.com/docs/mod_nss.html#Directives*

CURLOPT_SSL_SESSIONID_CACHE

Pass a long set to 0 to disable libcurl's use of SSL session-ID caching. Set this to 1 to enable it. By default all transfers are done using the cache. While nothing ever should get hurt by attempting to reuse SSL session-IDs, there seem to be broken SSL implementations in the wild that may require you to disable this in order for you to succeed. (Added in 7.16.0)

CURLOPT_SSL_OPTIONS

Pass a long with a bitmask to tell libcurl about specific SSL behaviors.

CURLSSLOPT_ALLOW_BEAST is the only supported bit and by setting this the user will tell libcurl to not attempt to use any work-arounds for a security flaw in the SSL3 and TLS1.0 protocols. If this option isn't used or this bit is set to 0, the SSL layer libcurl uses may use a work-around for this flaw although it might cause interoperability problems with some (older) SSL implementations. WARNING: avoiding this work-around loosens the security, and by setting this option to 1 you ask for exactly that. (Added in 7.25.0)

CURLOPT_KRBLEVEL

Pass a char * as parameter. Set the kerberos security level for FTP; this also enables kerberos 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 kerberos support for FTP.

(This option was known as CURLOPT_KRB4LEVEL up to 7.16.3)

CURLOPT_GSSAPI_DELEGATION

Set the parameter to CURLGSSAPI_DELEGATION_FLAG to allow unconditional GSSAPI credential delegation. The delegation is disabled by default since 7.21.7. Set the parameter to CURLGSSAPI_DELEGATION_POLICY_FLAG to delegate only if the OK-AS-DELEGATE flag is set in the service ticket in case this feature is supported by the GSSAPI implementation and the definition of GSS_C_DELEG_POLICY_FLAG was available at compile-time. (Added in 7.22.0)

SSH OPTIONS

CURLOPT_SSH_AUTH_TYPES

Pass a long set to a bitmask consisting of one or more of CURLSSH_AUTH_PUBLICKEY, CURLSSH_AUTH_PASSWORD, CURLSSH_AUTH_HOST, CURLSSH_AUTH_KEYBOARD. Set CURLSSH_AUTH_ANY to let libcurl pick one. Currently CURLSSH_AUTH_HOST has no effect. (Added in 7.16.1)

CURLOPT_SSH_HOST_PUBLIC_KEY_MD5

Pass a char * pointing to a string containing 32 hexadecimal digits. The string should be the 128 bit MD5 checksum of the remote host's public key, and libcurl will reject the connection to the host unless the md5sums match. This option is only for SCP and SFTP transfers. (Added in 7.17.1)

CURLOPT_SSH_PUBLIC_KEYFILE

Pass a char * pointing to a file name for your public key. If not used, libcurl defaults to **$HOME/.ssh/id_dsa.pub** if the HOME environment variable is set, and just "id_dsa.pub" in the current directory if HOME is not set. (Added in 7.16.1) If an empty string is passed, libcurl will pass no public key to libssh2 which then tries to compute it from the private key, this is known to work when libssh2 1.4.0+ is linked against OpenSSL. (Added in 7.26.0)

CURLOPT_SSH_PRIVATE_KEYFILE

Pass a char * pointing to a file name for your private key. If not used, libcurl defaults to **$HOME/.ssh/id_dsa** if the HOME environment variable is set, and just "id_dsa" in the current directory if HOME is not set. If the file is password-protected, set the password with *CURLOPT_KEYPASSWD*. (Added in 7.16.1)

CURLOPT_SSH_KNOWNHOSTS

Pass a pointer to a zero terminated string holding the file name of the known_host file to use. The known_hosts file should use the OpenSSH file format as supported by libssh2. If this file is specified, libcurl will only accept connections with hosts that are known and present in that file, with a matching public key. Use *CURLOPT_SSH_KEYFUNCTION* to alter the default behavior on host and key (mis)matching. (Added in 7.19.6)

CURLOPT_SSH_KEYFUNCTION

Pass a pointer to a curl_sshkeycallback function. It gets called when the known_host matching has been done, to allow the application to act and decide for libcurl how to proceed. The callback will only be called if *CURLOPT_SSH_KNOWNHOSTS* is also set.

The curl_sshkeycallback function gets passed the CURL handle, the key from the known_hosts file, the key from the remote site, info from libcurl on the matching status and a custom pointer (set with *CURLOPT_SSH_KEYDATA*). It MUST return one of the following return codes to tell libcurl how to act:

CURLKHSTAT_FINE_ADD_TO_FILE

The host+key is accepted and libcurl will append it to the known_hosts file before continuing with the connection. This will also add the host+key combo to the known_host pool kept in memory if it wasn't already present there. The adding of data to the file is done by completely replacing the file with a new copy, so the permissions of the file must allow this.

CURLKHSTAT_FINE

The host+key is accepted libcurl will continue with the connection. This will also add the host+key combo to the known_host pool kept in memory if it wasn't already present there.

CURLKHSTAT_REJECT

The host+key is rejected. libcurl will deny the connection to continue and it will be closed.

CURLKHSTAT_DEFER

The host+key is rejected, but the SSH connection is asked to be kept alive. This feature could be used when the app wants to somehow return back and act on the host+key situation and then retry without needing the overhead of setting it up from scratch again.

(Added in 7.19.6)

CURLOPT_SSH_KEYDATA

Pass a void * as parameter. This pointer will be passed along verbatim to the callback set with *CURLOPT_SSH_KEYFUNCTION*. (Added in 7.19.6)

OTHER OPTIONS

CURLOPT_PRIVATE

Pass a void * 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 in multiple threads, you **MUST** use the locking methods in the share handle. See *curl_share_setopt(3)* for details.

If you add a share that is set to share cookies, your easy handle will use that cookie cache and get the cookie engine enabled. If you unshare an object that was using cookies (or change to another object that doesn't share cookies), the easy handle will get its cookie engine disabled.

Data that the share object is not set to share will be dealt with the usual way, as if no share was used.

CURLOPT_NEW_FILE_PERMS

Pass a long as a parameter, containing the value of the permissions that will be assigned to newly created files on the remote server. The default value is *0644*, but any valid value can be used. The

only protocols that can use this are *sftp://*, *scp://*, and *file://*. (Added in 7.16.4)

CURLOPT_NEW_DIRECTORY_PERMS

Pass a long as a parameter, containing the value of the permissions that will be assigned to newly created directories on the remote server. The default value is *0755*, but any valid value can be used. The only protocols that can use this are *sftp://*, *scp://*, and *file://*. (Added in 7.16.4)

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)