NativeSPTransportOption

The `<TransportOption>` element allows implementation-specific properties to be passed into the SOAP transport implementation class, bypassing the usual abstractions.

> ⚠ This can be a powerful, but dangerous feature. Used improperly, crashes are quite possible to induce.

```
<TransportOption provider="CURL" option="100">2</TransportOption>
```

General Attributes

- `provider`(string)
    - Identifies the transport implementation the option is intended to be passed into. "CURL" is generally supported, and "OpenSSL" is supported in Version 2.4 and higher.

CURL Provider

When `provider` is set to "CURL", the following information applies.

Attributes

- `option`(string)
    - A CURL option number, which can be determined by examining the curl.h header file or the curl_easy_init man page. Find below a list of options:

```
[...]

NETWORK OPTIONS
        CURLOPT_URL (10002)
            The actual URL to deal with. The parameter should be a char * to
            a zero terminated string.

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

            The  string  given to CURLOPT_URL (10002) must be url-encoded and follow
            RFC 2396 (http://curl.haxx.se/rfc/rfc2396.txt).

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

            CURLOPT_URL (10002)  is  the  only  option  that  must  be  set  before
            curl_easy_perform(3) is called.

            CURLOPT_PROTOCOLS (181)  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 (181)
            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_PROTO-
            COLS. (Added in 7.19.4)

        CURLOPT_REDIR_PROTOCOLS (182)
            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_FOLLOWLO-
CATION 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 (10004)
   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 sepa-
   rate option. If not specified, libcurl will default to using
   port 1080 for proxies. CURLOPT_PROXY (10004)PORT.

   When you tell the library to use an HTTP proxy, libcurl will
   transparently convert operations to HTTP even if you specify an
   FTP URL etc. This may have an impact on what other features of
   the library you can use, such as CURLOPT_QUOTE (10028) and similar FTP
   specifics that don't work unless you tunnel through the HTTP
   proxy. Such tunneling is activated with CURLOPT_HTTPPROXYTUNNEL (61).

   libcurl respects the environment variables http_proxy,
   ftp_proxy, all_proxy etc, if any of those are set. The CUR-
   LOPT_PROXY option does however override any possibly set envi-
   ronment 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 vari-
   ables can be specified the exact same way as the proxy can be
   set with CURLOPT_PROXY (10004), include protocol prefix (http://) and
   embedded user + password.

CURLOPT_PROXYPORT (10004)
   Pass a long with this option to set the proxy port to connect to
   unless it is specified in the proxy string CURLOPT_PROXY (10004).

CURLOPT_PROXYTYPE (10004)
   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 (4), CURL-
   PROXY_SOCKS5, CURLPROXY_SOCKS4A (6) and CURL-
   PROXY_SOCKS5_HOSTNAME (7). The HTTP type is
   default. (Added in 7.10)

CURLOPT_NOPROXY (10177)
   Pass a pointer to a zero terminated string. The 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 (61)
   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 (10179)
   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

CURLOPT_SOCKS5_GSSAPI_NEC (180)
   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 ref-
   erence implementation does not. If enabled, this option allows
   the unprotected exchange of the protection mode negotiation.
   (Added in 7.19.4).

CURLOPT_INTERFACE (10062)
   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.

CURLOPT_LOCALPORT (139)
   Pass a long. This sets the local port number of the socket used
   for connection. This can be used in combination with CUR-
   LOPT_INTERFACE and you are recommended to use CURLOPT_LOCALPOR-
   TRANGE as well when this is set. Valid port numbers are 1 –
   65535. (Added in 7.15.2)

CURLOPT_LOCALPORT (139)RANGE
   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 (139) 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 fail-
   ures. (Added in 7.15.2)

CURLOPT_DNS_CACHE_TIMEOUT (92)
   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 (91)
   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 CUR-
   LOPT_SHARE and curl_share_init.

CURLOPT_BUFFERSIZE (98)
   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 (3)
   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 (121)
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 (171)
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)

NAMES and PASSWORDS OPTIONS (Authentication)
CURLOPT_NETRC (51)
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 (10002).

libcurl uses a user name (and supplied or prompted password) supplied with CURLOPT_USERPWD (10005) 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 (51)
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 (51) is set, libcurl will attempt to find a .netrc file in the current user's home directory. (Added in 7.10.9)

CURLOPT_USERPWD (10005)

Pass a char * as parameter, which should be [user name]:[pass-
word] to use for the connection. Use CURLOPT_HTTPAUTH (107) 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 (52), libcurl might per-
form 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 (105) 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 acciden-
tal information leakage.

CURLOPT_PROXY (10004)USERPWD
    Pass a char * as parameter, which should be [user name]:[pass-
    word] to use for the connection to the HTTP proxy. Use CUR-
    LOPT_PROXYAUTH to decide the authentication method.

CURLOPT_USERNAME (10173)
    Pass a char * as parameter, which should be pointing to the zero
    terminated user name to use for the transfer.

    CURLOPT_USERNAME (10173) sets the user name to be used in protocol
    authentication. You should not use this option together with the
    (older) CURLOPT_USERPWD (10005) option.

    In order to specify the password to be used in conjunction with
    the user name use the CURLOPT_PASSWORD (10174) option. (Added in
    7.19.1)

CURLOPT_PASSWORD (10174)
    Pass a char * as parameter, which should be pointing to the zero
    terminated password to use for the transfer.

    The CURLOPT_PASSWORD (10174) option should be used in conjunction with
    the CURLOPT_USERNAME (10173) option. (Added in 7.19.1)

CURLOPT_PROXY (10004)USERNAME
    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_PROXY (10004)USERNAME option should be used in same way as
    the CURLOPT_PROXY (10004)USERPWD is used. In comparison to CUR-
    LOPT_PROXYUSERPWD the CURLOPT_PROXY (10004)USERNAME allows the username
    to contain a colon, like in the following example:
    "sip:user@example.com". The CURLOPT_PROXY (10004)USERNAME 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_PROX-
    YUSERPWD option.

    In order to specify the password to be used in conjunction with
    the user name use the CURLOPT_PROXY (10004)PASSWORD option. (Added in
    7.19.1)

CURLOPT_PROXY (10004)PASSWORD
    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_PROXY (10004)PASSWORD option should be used in conjunction
    with the CURLOPT_PROXY (10004)USERNAME option. (Added in 7.19.1)

CURLOPT_HTTPAUTH (107)
    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 (10005) option or with the CURLOPT_USERNAME (10173) and the CURLOPT_USERPASSWORD 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 sup-ported 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 authen-tication over public networks than the regular old-fash-ioned 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 sim-ply 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 OpenSSL support for this option to work, or build libcurl on Windows.

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

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

CURLOPT_PROXY (10004)AUTH
   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_PROXY (10004)USERPWD

option. The bitmask can be constructed by or'ing together the
bits listed above for the CURLOPT_HTTPAUTH (107) option. As of this
writing, only Basic, Digest and NTLM work. (Added in 7.10.7)

HTTP OPTIONS
    CURLOPT_AUTOREFERER (58)
       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_ENCODING (10102)
       Sets the contents of the Accept-Encoding: header sent in an HTTP
       request, and enables decoding of a response when a Content-
       Encoding: header is received. Three encodings are supported:
       identity, which does nothing, deflate which requests the server
       to compress its response using the zlib algorithm, and gzip
       which requests the gzip algorithm. If a zero-length string is
       set, then an Accept-Encoding: header containing all supported
       encodings is sent.

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

    CURLOPT_FOLLOWLOCATION (52)
       A parameter set to 1 tells the library to follow any Location:
       header that the server sends as part of an HTTP header.

       This means that the library will re-send the same request on the
       new location and follow new Location: headers all the way until
       no more such headers are returned. CURLOPT_MAXREDIRS (68) can be used
       to limit the number of redirects libcurl will follow.

       Since 7.19.4, libcurl can limit what protocols it will automati-
       cally follow. The accepted protocols are set with CUR-
       LOPT_REDIR_PROTOCOLS and it excludes the FILE protocol by
       default.

    CURLOPT_UNRESTRICTED_AUTH (105)
       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 set-
       ting CURLOPT_FOLLOWLOCATION (52).

    CURLOPT_MAXREDIRS (68)
       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 (52) 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_POST (47)REDIR
       Pass a bitmask to control how libcurl acts on redirects after
       POSTs that get a 301 or 302 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. 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 CUR-
       LOPT_FOLLOWLOCATION. (Added in 7.17.1) (This option was known
       as CURLOPT_POST (47)301 up to 7.19.0 as it only supported the 301 way
       before then)

CURLOPT_PUT (54)
   A parameter set to 1 tells the library to use HTTP PUT to trans-
   fer data. The data should be set with CURLOPT_READDATA (10009) and CUR-
   LOPT_INFILESIZE.

   This option is deprecated and starting with version 7.12.1 you
   should instead use CURLOPT_UPLOAD (46).

CURLOPT_POST (47)
   A parameter set to 1 tells the library to do a regular HTTP
   post. This will 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_POST (47)FIELDS (10015) or CURLOPT_COPYPOSTFIELDS (10165) options
   to specify what data to post and CURLOPT_POST (47)FIELDS (10015)IZE or CUR-
   LOPT_POSTFIELDSIZE_LARGE to set the data size.

   Optionally, you can provide data to POST using the CURLOPT_READ-
   FUNCTION and CURLOPT_READDATA (10009) options but then you must make
   sure to not set CURLOPT_POST (47)FIELDS (10015) to anything but NULL. When
   providing data with a callback, you must transmit it using chun-
   ked transfer-encoding or you must set the size of the data with
   the CURLOPT_POST (47)FIELDS (10015)IZE or CURLOPT_POST (47)FIELDS (10015)IZE_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 set-
   ting your own with CURLOPT_HTTPHEADER (10023).

   Using POST with HTTP 1.1 implies the use of a "Expect: 100-con-
   tinue" header. You can disable this header with CURLOPT_HTTP-
   HEADER 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 (10023). With HTTP 1.0 or
   without chunked transfer, you must specify the size in the
   request.

   When setting CURLOPT_POST (47) to 1, it will automatically set CUR-
   LOPT_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 (44) or CURLOPT_HTTPGET (80) or similar.

CURLOPT_POST (47)FIELDS (10015)
   Pass a void * as parameter, which should be the full data to
   post in an HTTP POST operation. You must make sure that the data
   is formatted the way you want the server to receive it. libcurl
   will not convert or encode it for you. Most web servers will
   assume this data to be url-encoded.

   The pointed data are NOT copied by the library: as a conse-
   quence, 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 (47). Using CURLOPT_POST (47)FIELDS (10015)
   implies CURLOPT_POST (47).

   If you want to do a zero-byte POST, you need to set CUR-
   LOPT_POSTFIELDSIZE explicitly to zero, as simply setting CUR-
   LOPT_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-con-
tinue" header.   You can disable this header with CURLOPT_HTTP-
HEADER as usual.

To make multipart/formdata posts (aka RFC2388-posts), check out
the CURLOPT_HTTPPOST (10024) option.

CURLOPT_POST (47)FIELDS (10015)IZE
   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_POST (47)FIELDS (10015)IZE_LARGE
   Pass a curl_off_t as parameter. Use this to set the size of the
   CURLOPT_POST (47)FIELDS (10015) data to prevent libcurl from doing strlen()
   on the data to figure out the size. This is the large file ver-
   sion of the CURLOPT_POST (47)FIELDS (10015)IZE option. (Added in 7.11.1)

CURLOPT_COPYPOSTFIELDS (10165)
   Pass a char * as parameter, which should be the full data to
   post in an HTTP POST operation. It behaves as the CURLOPT_POST (47)-
   FIELDS 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_POST (47)FIELDS (10015)IZE or CUR-
   LOPT_POSTFIELDSIZE_LARGE: If the size has not been set prior to
   CURLOPT_COPYPOSTFIELDS (10165), the data are assumed to be a NUL-termi-
   nated 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 (10165), unless   another   CUR-
   LOPT_POSTFIELDS or CURLOPT_COPYPOSTFIELDS (10165) option is issued.
   (Added in 7.17.1)

CURLOPT_HTTPPOST (10024)
   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-con-
   tinue" header. You can disable this header with CURLOPT_HTTP-
   HEADER as usual.

   When setting CURLOPT_HTTPPOST (10024), it will automatically set CUR-
   LOPT_NOBODY to 0 (since 7.14.1).

CURLOPT_REFERER (10016)
   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 (10023).

CURLOPT_USERAGENT (10018)
   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_HTTP-
   HEADER.

CURLOPT_HTTPHEADER (10023)
   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 (10022), CURLOPT_USERAGENT (10018) and CURLOPT_REFERER (10016).

CURLOPT_HTTP200ALIASES (10104)
    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 (10022)
    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 explictly 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_COOKIE (10022)FILE
    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 han-

dle, 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_COOKIE (10022)JAR
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 (41) or CUR-
LOPT_DEBUGFUNCTION will get a warning to display, but that is
the only visible feedback you get about this possibly lethal
situation.

CURLOPT_COOKIE (10022)SESSION
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, indepen-
dent 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_COOKIE (10022)LIST
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 speci-
fied by CURLOPT_COOKIE (10022)JAR. (Added in 7.17.1)

CURLOPT_HTTPGET (80)
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 (80) to 1, it will automatically set
CURLOPT_NOBODY (44) to 0 (since 7.14.1).

CURLOPT_HTTP_VERSION (84)
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 (136)
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 (158)
           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 (10102) for that. (added in 7.16.2)

       CURLOPT_HTTP_TRANSFER_DECODING (157)
           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)

TFTP OPTIONS
       CURLOPT_TFTP_BLKSIZE (178)
           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 (10017)
           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 instruc-
           tion. 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,  option-
           ally  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 avail-
           able, libcurl will report CURLE_FTP_PORT_FAILED for the  handle.
           Invalid  port/range  settings  are  ignored.  IPv6 addresses fol-
           lowed 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 (10028)
           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 com-
           mand 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.  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_POST (47)QUOTE (10039)
           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 (10028). Disable this operation again by
setting a NULL to this option.

CURLOPT_PREQUOTE (10093)
   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 (10028). Disable this operation again
   by setting a NULL to this option. Before version 7.15.6, if you
   also set CURLOPT_NOBODY (44) to 1, this option didn't work.

CURLOPT_DIRLISTONLY (48)
   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 neces-
   sary.

   Do NOT use this option if you also use CURLOPT_WILDCARDMATCH (197) as
   it will effectively break that feature then.

   (This option was known as CURLOPT_FTPLISTONLY up to 7.16.4)

CURLOPT_APPEND (50)
   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 (106)
   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 (10017)). 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 (85)
   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 (188)
   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 (110)
   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 (112)
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 (13). It is recom-
mended that if used in conjunction with CURLOPT_TIMEOUT (13), you set
CURLOPT_FTP_RESPONSE_TIMEOUT (112) to   a value smaller than   CUR-
LOPT_TIMEOUT. (Added in 7.10.8)

CURLOPT_FTP_ALTERNATIVE_TO_USER (10147)
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 (137)
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_USE_SSL (119)
Pass a long using one of the values from below, to make libcurl
use your desired level of SSL for the FTP transfer. (Added in
7.11.0)

(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_FTPSSLAUTH (129)
          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 (119)). (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 (154)
          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 unen-
          crypted. 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 (10134)
          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 (138)
          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 com-
                mands. 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 multi-
                cwd case). This is somewhat more standards compliant than
                'nocwd' but without the full penalty of 'multicwd'.
          (Added in 7.15.1)

PROTOCOL OPTIONS
       CURLOPT_TRANSFERTEXT (53)
          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 simi-
lar.

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 per-
forms a standard transfer.

CURLOPT_PROXY (10004)_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 an 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 (53) has no effect when doing FTP via a proxy.
Beware that not all proxies support this feature. (Added in
7.18.0)

CURLOPT_CRLF (27)
Convert Unix newlines to CRLF newlines on transfers.

CURLOPT_RANGE (10007)
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 multi-
ple 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 Sec-
tion 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 (21)
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 con-
tinue 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 (21)
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 (10036)
Pass a pointer to a zero terminated string as parameter. It will
be used instead of GET or HEAD when doing an HTTP request, or
instead of LIST or NLST when doing a FTP directory listing. This
is useful for doing DELETE or other more or less obscure HTTP
requests. Don't do this at will, make sure your server supports
the command first.

When you change the request method by setting CURLOPT_CUSTOMRE-
QUEST 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 (10036) 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 (44), to switch to a
proper POST, use CURLOPT_POST (47) or CURLOPT_POST (47)FIELDS (10015) 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 con-
tents. 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 (47) and CURLOPT_POST (47)FIELDS (10015)
to set POST data. Use CURLOPT_HTTPHEADER (10023) to replace or extend
the set of headers sent by libcurl. Use CURLOPT_HTTP_VERSION (84) to
change HTTP version.

CURLOPT_FILETIME (10001)
    Pass a long. If it is 1, libcurl will attempt to get the modifi-
    cation 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 (44)
    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 (80). Change
    request to POST with CURLOPT_POST (47) etc.

CURLOPT_INFILESIZE (10009)
    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_INFILE (10009)-
    SIZE_LARGE.

    For uploading using SCP, this option or CURLOPT_INFILE (10009)SIZE_LARGE
    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_INFILESIZE_LARGE (10009)
    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_INFILE (10009)SIZE 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 (46)
    A parameter set to 1 tells the library to prepare for an upload.
    The CURLOPT_READDATA (10009) and CURLOPT_INFILESIZE (10009) or
CURLOPT_INFILESIZE_LARGE (10009)-
    options are also interesting for uploads. If the pro-
    tocol 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-con-
    tinue" header. You can disable this header with CURLOPT_HTTP-
    HEADER 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 (10023). With HTTP 1.0 or without chunked transfer, you must specify the size.

CURLOPT_MAXFILESIZE (114)
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 (114)
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 (33)
Pass a long as parameter. This defines how the CURLOPT_TIMEVALUE (34) time value is treated. You can set this parameter to CURL_TIME-COND_IFMODSINCE or CURL_TIMECOND_IFUNMODSINCE. This feature applies to HTTP, FTP, and RTSP.

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_get-info(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 (34)
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 (33).

CONNECTION OPTIONS
CURLOPT_TIMEOUT (13)
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 opera-tions 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 (99) is set.

CURLOPT_TIMEOUT_MS (13)
Like CURLOPT_TIMEOUT (13) 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 resolu-tion for timeouts with a minimum timeout allowed of one second. (Added in 7.16.2)

CURLOPT_LOW_SPEED_LIMIT (19)
Pass a long as parameter. It contains the transfer speed in bytes per second that the transfer should be below during CUR-LOPT_LOW_SPEED_TIME seconds for the library to consider it too slow and abort.

CURLOPT_LOW_SPEED_TIME (20)
Pass a long as parameter. It contains the time in seconds that

the transfer should be below the CURLOPT_LOW_SPEED_LIMIT (19) for the
library to consider it too slow and abort.

CURLOPT_MAX_SEND_SPEED_LARGE (30145)
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 (30146)
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 (71)
Pass a long. The set number will be the persistent connection
cache size. The set amount will be the maximum amount of simul-
taneously 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 connec-
tions 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 (72)
(Obsolete) This option does nothing.

CURLOPT_FRESH_CONNECT (74)
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 (75)
Pass a long. Set to 1 to make the next transfer explicitly close
the connection when done. Normally, libcurl keeps all connec-
tions 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 (78)
Pass a long. It should contain the maximum time in seconds that
you allow the connection to the server to take. This only lim-
its the connection phase, once it has connected, this option is
of no more use. Set to zero to disable connection timeout (it
will then only timeout on the system's internal timeouts). See
also the CURLOPT_TIMEOUT (13) option.

In unix-like systems, this might cause signals to be used unless
CURLOPT_NOSIGNAL (99) is set.

CURLOPT_CONNECTTIMEOUT_MS (78)
Like CURLOPT_CONNECTTIMEOUT (78) 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 (113)
                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 (1)
                     Resolve to IPv4 addresses.

                CURL_IPRESOLVE_V6 (2)
                     Resolve to IPv6 addresses.

       CURLOPT_CONNECT_ONLY (141)
                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)

SSL and SECURITY OPTIONS
       CURLOPT_SSLCERT (10025)
                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_SSLCERT (10025)TYPE.

                With  NSS  this  is  the nickname of the certificate you wish to
                authenticate with.

       CURLOPT_SSLCERTTYPE (10025)
                Pass a pointer to a zero terminated  string  as  parameter.  The
                string  should be the format of your certificate. Supported for-
                mats are "PEM" and "DER".  (Added in 7.9.3)

       CURLOPT_SSLKEY (10087)
                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_SSLKEY (10087)TYPE.

       CURLOPT_SSLKEYTYPE (10087)
                Pass a pointer to a zero terminated  string  as  parameter.  The
                string  should be the format of your private key. Supported for-
                mats are "PEM", "DER" and "ENG".

                The format "ENG" enables you to load  the  private  key  from  a
                crypto engine. In this case CURLOPT_SSLKEY (10087) is used as an identi-
                fier passed to the engine. You have to  set  the  crypto  engine
                with  CURLOPT_SSLENGINE (10089).  "DER"  format key file currently does
                not work because of a bug in OpenSSL.

       CURLOPT_KEYPASSWD (10026)
                Pass a pointer to a zero terminated string as parameter. It will
                be  used  as  the password required to use the CURLOPT_SSLKEY (10087) or
                CURLOPT_SSH_PRIVATE_KEYFILE (10153) 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_SSLKEY (10087)PASSWD up to 7.16.4  and
                CURLOPT_SSLCERT (10025)PASSWD up to 7.9.2)
```

CURLOPT_SSLENGINE (10089)
   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 (10089)
   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 con-
   figurations 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 (32)
   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 (64)
   Pass a long as parameter.

   This option determines whether curl verifies the authenticity of
   the peer's certificate. A value of 1 means curl verifies; zero
   means it doesn't. The default is nonzero, but before 7.10, it
   was zero.

   When negotiating an SSL connection, the server sends a certifi-
   cate indicating its identity. Curl verifies whether the cer-
   tificate is authentic, i.e. that you can trust that the server
   is who the certificate says it is. This trust is based on a
   chain of digital signatures, rooted in certification authority
   (CA) certificates you supply.   As of 7.10, curl installs a
   default bundle of CA certificates and you can specify alternate
   certificates with the CURLOPT_CAINFO (10065) option or the CURLOPT_CAP-
   ATH option.

   When CURLOPT_SSL_VERIFYPEER (64) is nonzero, and the verification
   fails to prove that the certificate is authentic, the connection
   fails. When the option is zero, the connection succeeds regard-
   less.

   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 (81) to control that.

CURLOPT_CAINFO (10065)
   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_VERI-
FYPEER   option.   If   CURLOPT_SSL_VERIFYPEER (64) is zero, CUR-
LOPT_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 cer-
tificate database resides in.

CURLOPT_ISSUERCERT (10170)
    Pass a char * to a zero terminated string naming a file holding
    a CA certificate in PEM format. If the option is set, an addi-
    tional 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 cer-
    tificate is from a specific branch of the tree.

    This option makes sense only when used in combination with the
    CURLOPT_SSL_VERIFYPEER (64) 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 ses-
    sion has failed due to a mismatch with the issuer of peer cer-
    tificate (CURLOPT_SSL_VERIFYPEER (64) has to be set too for the check
    to fail). (Added in 7.19.0)

CURLOPT_CAPATH (10097)
    Pass a char * to a zero terminated string naming a directory
    holding multiple CA certificates to verify the peer with. The
    certificate directory must be prepared using the openssl
    c_rehash utility. This makes sense only when used in combination
    with the CURLOPT_SSL_VERIFYPEER (64) option. If CURLOPT_SSL_VERI-
    FYPEER is zero, CURLOPT_CAPATH (10097) need not even indicate an acces-
    sible path.   The CURLOPT_CAPATH (10097) 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.

CURLOPT_CRLFILE (27)
    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 cer-
    tificate chain if a CRL file is passed.

    This option makes sense only when used in combination with the
    CURLOPT_SSL_VERIFYPEER (64) 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 verifi-
    cation due to a revocation information found in the CRL does not
    trigger this specific error. (Added in 7.19.0)

CURLOPT_CERTINFO (172)
    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 (10076)
   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 (10077)
   Pass a char * to the zero terminated path name to the Entropy
   Gathering Daemon socket. It will be used to seed the random
   engine for SSL.

CURLOPT_SSL_VERIFYHOST (81)
   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 certifi-
   cate indicating its identity.

   When CURLOPT_SSL_VERIFYHOST (81) 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 con-
   nect.

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

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

   The default, since 7.10, is 2.

   This option controls checking the server's claimed identity.
   The server could be lying.   To control lying, see   CUR-
   LOPT_SSL_VERIFYPEER.

CURLOPT_SSL_CIPHER_LIST (10083)
   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 sepa-
   rators 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 (150)
   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_KRBLEVEL (10063)
                        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)

        [...]
```

Element Content

The content of the element is an option-specific string that depends on the option being set.

Examples

Restrict transport to IPv4

```
<TransportOption provider="CURL" option="113">1</TransportOption>
```

Set transport timeout in seconds

```
<TransportOption provider="CURL" option="13">120</TransportOption>
```

Set HTTP proxy address

```
<TransportOption provider="CURL" option="10004">proxy.example.org:8080</TransportOption>
```

Verify SSL certificate

```
<!-- Known to work with curl 7.26 and GnuTLS 2.12.20 (Debian 7.0) -->
<TransportOption provider="CURL" option="64">1</TransportOption>
<TransportOption provider="CURL" option="81">2</TransportOption>
<TransportOption provider="CURL" option="10065">/etc/ssl/certs/ca-certificates.crt</TransportOption>
```

OpenSSL Provider (Version 2.4 and Above)

When provider is set to "OpenSSL", the following information applies.

Attributes

- option(string)
  - The only supported value is currently "SSL_OP_ALLOW_UNSAFE_LEGACY_RENEGOTIATION". If the element content is "1" or "true", this causes the option for unsafe renegotiation to be applied, allowing interop with older servers when using very recent versions of OpenSSL.

Element Content

The content of the element is an option-specific string that depends on the option being set.