NAME

curl_easy_pause - pause and unpause a connection

SYNOPSIS

#include <curl/curl.h>

CURLcode curl_easy_pause(CURL *****handle*, int** *bitmask* **);**

DESCRIPTION

Using this function, you can explicitly mark a running connection to get paused, and you can unpause a connection that was previously paused.

A connection can be paused by using this function or by letting the read or the write callbacks return the proper magic return code (*CURL_READFUNC_PAUSE* and *CURL_WRITEFUNC_PAUSE*). A write callback that returns pause signals to the library that it couldn't take care of any data at all, and that data will then be delivered again to the callback when the writing is later unpaused.

While it may feel tempting, take care and notice that you cannot call this function from another thread. To unpause, you may for example call it from the progress callback (*CURLOPT_PROGRESSFUNCTION(3)*), which gets called at least once per second, even if the connection is paused.

When this function is called to unpause reading, the chance is high that you will get your write callback called before this function returns.

The **handle** argument is of course identifying the handle that operates on the connection you want to pause or unpause.

The **bitmask** argument is a set of bits that sets the new state of the connection. The following bits can be used:

CURLPAUSE_RECV
: Pause receiving data. There will be no data received on this connection until this function is called again without this bit set. Thus, the write callback (*CURLOPT_WRITEFUNCTION(3)*) won't be called.

CURLPAUSE_SEND
: Pause sending data. There will be no data sent on this connection until this function is called again without this bit set. Thus, the read callback (*CURLOPT_READFUNCTION(3)*) won't be called.

CURLPAUSE_ALL
: Convenience define that pauses both directions.

CURLPAUSE_CONT
: Convenience define that unpauses both directions.

RETURN VALUE

CURLE_OK (zero) means that the option was set properly, and a non-zero return code means something wrong occurred after the new state was set. See the *libcurl-errors(3)* man page for the full list with descriptions.

LIMITATIONS

The pausing of transfers does not work with protocols that work without network connectivity, like FILE://. Trying to pause such a transfer, in any direction, will cause problems in the worst case or an error in the best case.

AVAILABILITY

This function was added in libcurl 7.18.0. Before this version, there was no explicit support for pausing transfers.

USAGE WITH THE MULTI-SOCKET INTERFACE

Before libcurl 7.32.0, when a specific handle was unpaused with this function, there was no particular forced rechecking or similar of the socket's state, which made the continuation of the transfer get delayed until next multi-socket call invoke or even longer. Alternatively, the user could forcibly call for example curl_multi_socket_all(3) - with a rather hefty performance penalty.

Starting in libcurl 7.32.0, unpausing a transfer will schedule a timeout trigger for that handle 1 millisecond into the future, so that a curl_multi_socket_action(... CURL_SOCKET_TIMEOUT) can be used immediately afterwards to get the transfer going again as desired.

MEMORY USE

When pausing a read by returning the magic return code from a write callback, the read data is already in libcurl's internal buffers so it'll have to keep it in an allocated buffer until the reading is again unpaused using this function.

If the downloaded data is compressed and is asked to get uncompressed automatically on download, libcurl will continue to uncompress the entire downloaded chunk and it will cache the data uncompressed. This has the side- effect that if you download something that is compressed a lot, it can result in a very large data amount needing to be allocated to save the data during the pause. This said, you should probably consider not using paused reading if you allow libcurl to uncompress data automatically.

SEE ALSO

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