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. Unlike most other libcurl functions, *curl_easy_pause(3)* can be used from within callbacks.

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 could not take care of any data at all, and that data is then delivered again to the callback when the transfer is 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)*).

When this function is called to unpause receiving, the write callback might get called before this function returns to deliver cached content. When libcurl delivers such cached data to the write callback, it is delivered as fast as possible, which may overstep the boundary set in *CURLOPT_MAX_RECV_SPEED_LARGE(3)* etc.

The **handle** argument identifies the transfer you want to pause or unpause.

A paused transfer is excluded from low speed cancels via the *CURLOPT_LOW_SPEED_LIMIT(3)* option and unpausing a transfer resets the time period required for the low speed limit to be met.

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 is no data received on this connection until this function is called again without this bit set. Thus, the write callback (*CURLOPT_WRITEFUNCTION(3)*) is not called.

CURLPAUSE_SEND

Pause sending data. There is no data sent on this connection until this function is called again without this bit set. Thus, the read callback (*CURLOPT_READFUNCTION(3)*) is not called.

CURLPAUSE_ALL

Convenience define that pauses both directions.

CURLPAUSE_CONT

Convenience define that unpauses both directions.

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, might cause problems or error.

MULTIPLEXED

When a connection is used multiplexed, like for HTTP/2, and one of the transfers over the connection is paused and the others continue flowing, libcurl might end up buffering contents for the paused transfer. It has to do this because it needs to drain the socket for the other transfers and the already announced window size for the paused transfer allows the server to continue sending data up to that window size amount. By default, libcurl announces a 32 megabyte window size, which thus can make libcurl end up buffering 32 megabyte of data for a paused stream.

When such a paused stream is unpaused again, any buffered data is delivered first.

EXAMPLE

```
int main(void)
{
  CURL *curl = curl_easy_init();
  if(curl) {
    /* pause a transfer in both directions */
    curl_easy_pause(curl, CURL_READFUNC_PAUSE | CURL_WRITEFUNC_PAUSE);

  }
}
```

MEMORY USE

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

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

If the download is done with HTTP/2 or HTTP/3, there is up to a stream window size worth of data that curl cannot stop but instead needs to cache while the transfer is paused. This means that if a window size of 64 MB is used, libcurl might end up having to cache 64 MB of data.

AVAILABILITY

Added in 7.18.0.

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.

SEE ALSO

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