NAME

CURLOPT_READFUNCTION − read callback for data uploads

SYNOPSIS

```
#include <curl/curl.h>

size_t read_callback(char *buffer, size_t size, size_t nitems, void *userdata);

CURLcode curl_easy_setopt(CURL *handle, CURLOPT_READFUNCTION, read_callback);
```

DESCRIPTION

Pass a pointer to your callback function, as the prototype shows above.

This callback function gets called by libcurl as soon as it needs to read data in order to send it to the peer − like if you ask it to upload or post data to the server. The data area pointed at by the pointer *buffer* should be filled up with at most *nitems* number of bytes by your function. *size* is always 1.

Set the *userdata* argument with the *CURLOPT_READDATA(3)* option.

Your function must return the actual number of bytes that it stored in the data area pointed at by the pointer *buffer*. Returning 0 signals end−of−file to the library and causes 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 have said you would 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 is not sent.

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.

The callback can return *CURL_READFUNC_PAUSE* to cause reading from this connection to pause. 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 is considered the final packet by the server end and the transfer ends there.

If you set this callback pointer to NULL, or do not set it at all, the default internal read function is used. It is doing an fread() on the FILE * userdata set with *CURLOPT_READDATA(3)*.

You can set the total size of the data you are sending by using *CURLOPT_INFILESIZE_LARGE(3)* or *CURLOPT_POSTFIELDSIZE_LARGE(3)*, depending on the type of transfer. For some transfer types it may be required and it allows for better error checking.

When this option is used in combination with telling libcurl to follow redirects with *CURLOPT_FOLLOWLOCATION(3)*, the data might need to be rewound and sent again. The *CURLOPT_SEEKFUNCTION(3)* can then be invoked for that rewind operation.

DEFAULT

fread(3)

PROTOCOLS

This functionality affects all supported protocols

EXAMPLE

```
size_t read_callback(char *ptr, size_t size, size_t nmemb, void *userdata)
{
  FILE *readhere = (FILE *)userdata;
  curl_off_t nread;
```

```
 /* copy as much data as possible into the 'ptr' buffer, but no more than
    'size' * 'nmemb' bytes. */
 size_t retcode = fread(ptr, size, nmemb, readhere);

 nread = (curl_off_t)retcode;

 fprintf(stderr, "*** We read %" CURL_FORMAT_CURL_OFF_T
         " bytes from file\n", nread);
 return retcode;
}

int main(int argc, char **argv)
{
 FILE *file = fopen(argv[1], "rb");
 CURLcode result;

 CURL *curl = curl_easy_init();
 if(curl) {
  /* set callback to use */
  curl_easy_setopt(curl, CURLOPT_READFUNCTION, read_callback);

  /* pass in suitable argument to callback */
  curl_easy_setopt(curl, CURLOPT_READDATA, (void *)file);

  result = curl_easy_perform(curl);
 }
}
```

HISTORY

CURL_READFUNC_PAUSE return code was added in 7.18.0 and CURL_READFUNC_ABORT was added in 7.12.1.

AVAILABILITY

Added in curl 7.1

RETURN VALUE

This returns CURLE_OK.

SEE ALSO

CURLOPT_POST(3), **CURLOPT_READDATA**(3), **CURLOPT_SEEKFUNCTION**(3), **CURLOPT_UPLOAD**(3), **CURLOPT_UPLOAD_BUFFERSIZE**(3), **CURLOPT_WRITEFUNCTION**(3)