NAME

curl_easy_perform - perform a blocking file transfer

SYNOPSIS

#include <curl/curl.h>

CURLcode curl_easy_perform(CURL **easy_handle*);

DESCRIPTION

Invoke this function after *curl_easy_init(3)* and all the *curl_easy_setopt(3)* calls are made, and will perform the transfer as described in the options. It must be called with the same **easy_handle** as input as the *curl_easy_init(3)* call returned.

curl_easy_perform(3) performs the entire request in a blocking manner and returns when done, or if it failed. For non-blocking behavior, see *curl_multi_perform(3)*.

You can do any amount of calls to *curl_easy_perform(3)* while using the same **easy_handle**. If you intend to transfer more than one file, you are even encouraged to do so. libcurl will then attempt to re-use the same connection for the following transfers, thus making the operations faster, less CPU intense and using less network resources. Just note that you will have to use *curl_easy_setopt(3)* between the invokes to set options for the following curl_easy_perform.

You must never call this function simultaneously from two places using the same **easy_handle**. Let the function return first before invoking it another time. If you want parallel transfers, you must use several curl easy_handles.

While the **easy_handle** is added to a multi handle, it cannot be used by *curl_easy_perform(3)*.

RETURN VALUE

CURLE_OK (0) means everything was ok, non-zero means an error occurred as <*curl/curl.h*> defines - see *libcurl-errors(3)*. If the **CURLOPT_ERRORBUFFER(3)** was set with *curl_easy_setopt(3)* there will be a readable error message in the error buffer when non-zero is returned.

EXAMPLE

```
CURL *curl = curl_easy_init();
if(curl) {
  CURLcode res;
  curl_easy_setopt(curl, CURLOPT_URL, "http://example.com");
  res = curl_easy_perform(curl);
  curl_easy_cleanup(curl);
}}
```

SEE ALSO

curl_easy_init(3), **curl_easy_setopt**(3), **curl_multi_add_handle**(3), **curl_multi_perform**(3), **libcurl-errors**(3),