NAME

curl_multi_info_read − read multi stack information

SYNOPSIS

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

CURLMsg *curl_multi_info_read(CURLM *multi_handle, int *msgs_in_queue);
```

DESCRIPTION

Ask the multi handle if there are any messages from the individual transfers. Messages may include information such as an error code from the transfer or the fact that a transfer is completed. More details on these should be written down as well.

Repeated calls to this function returns a new struct each time, until a NULL is returned as a signal that there is no more to get at this point. The integer pointed to with *msgs_in_queue* contains the number of remaining messages after this function was called.

When you fetch a message using this function, it is removed from the internal queue so calling this function again does not return the same message again. It instead returns new messages at each new invoke until the queue is emptied.

WARNING: The data the returned pointer points to does not survive calling *curl_multi_cleanup(3)*, *curl_multi_remove_handle(3)* or *curl_easy_cleanup(3)*.

The *CURLMsg* struct is simple and only contains basic information. If more involved information is wanted, the particular "easy handle" is present in that struct and can be used in subsequent regular *curl_easy_getinfo(3)* calls (or similar):

```
struct CURLMsg {
  CURLMSG msg;       /* what this message means */
  CURL *easy_handle; /* the handle it concerns */
  union {
    void *whatever;    /* message-specific data */
    CURLcode result;   /* return code for transfer */
  } data;
};
```

When **msg** is *CURLMSG_DONE*, the message identifies a transfer that is done, and then **result** contains the return code for the easy handle that completed.

At this point, there are no other **msg** types defined.

PROTOCOLS

This functionality affects all supported protocols

EXAMPLE

```
int main(void)
{
  CURLM *multi = curl_multi_init();
  CURL *curl = curl_easy_init();
  if(curl) {
    struct CURLMsg *m;

    /* call curl_multi_perform or curl_multi_socket_action first, then loop
       through and check if there are any transfers that have completed */

    do {
```

```
    int msgq = 0;
    m = curl_multi_info_read(multi, &msgq);
    if(m && (m->msg == CURLMSG_DONE)) {
      CURL *e = m->easy_handle;
      /* m->data.result holds the error code for the transfer */
      curl_multi_remove_handle(multi, e);
      curl_easy_cleanup(e);
    }
  } while(m);
 }
}
```

AVAILABILITY

Added in curl 7.9.6

RETURN VALUE

A pointer to a filled−in struct, or NULL if it failed or ran out of structs. It also writes the number of messages left in the queue (after this read) in the integer the second argument points to.

SEE ALSO

curl_multi_cleanup(3), **curl_multi_init**(3), **curl_multi_perform**(3)