NAME

CURLMOPT_NOTIFYFUNCTION − callback receiving notifications

SYNOPSIS

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

void notify_callback(CURLM *multi,     /* multi handle */
                     unsigned int notification, /* notification type */
                     CURL *easy,       /* easy handle */
                     void *notifyp);   /* private notify pointer */

CURLMcode curl_multi_setopt(CURLM *handle, CURLMOPT_NOTIFYFUNCTION, notify_callback);
```

DESCRIPTION

Pass a pointer to your callback function, which should match the prototype shown above.

When the multi handle processes transfers, changes can be observed by receiving notifications about them. This can eliminate the need to constantly interrogate the multi handle to observe such changes to act on them.

Notifications are collected and dispatched to the application's callback function at an appropriate time.

The notify callback is different from other callbacks in that it can use more libcurl API functions. Apart from *curl_multi_perform(3)*, *curl_multi_socket(3)*, *curl_multi_socket_action(3)*, *curl_multi_socket_all(3)* and *curl_multi_cleanup(3)* it may call all other methods on the multi and easy handles. This includes adding and removing easy handles to/from the multi handle.

This callback may get invoked at any time when interacting with libcurl. This may even happen after all transfers are done and *may also* happen *during* a call to *curl_multi_cleanup(3)* when cached connections are shut down.

CALLBACK ARGUMENTS

multi identifies the multi handle that triggered the notification.

notification is the type of notification, e.g. what happened. The following types are available right now. In the future, new ones might be added.

CURLMNOTIFY_INFO_READ

When enabled via *curl_multi_notify_enable(3)*, this informs the application that there are new messages to be processed via *curl_multi_info_read(3)*.

This notification happens whenever a message is added to an empty message stack in the multi handle and not for subsequent additions. The notification callback is then expected to read all available message, emptying the stack, so a subsequent addition triggers the notification again.

The *easy* handle passed is an internal handle.

CURLMNOTIFY_EASY_DONE

When enabled via *curl_multi_notify_enable(3)*, this notification is triggered when an easy handle has finished. This happens both for successful and failed transfers.

The *easy* handle passed is the transfer that is done. This *may* be an internal handle when DoH or other features are used.

easy identifies the transfer involved. This may be one of the application's own easy handle or an internal handle.

notifyp is set with *CURLMOPT_NOTIFYDATA(3)*.

DEFAULT

NULL (no callback)

PROTOCOLS

This functionality affects all supported protocols

EXAMPLE

```
struct priv {
  void *ours;
};

static void notify_cb(CURLM *multi, unsigned int notification,
                      CURL *easy, void *notifyp)
{
  struct priv *p = notifyp;
  printf("my ptr: %p\n", p->ours);
  /* ... */
}

int main(void)
{
  struct priv setup;
  CURLM *multi = curl_multi_init();
  /* ... use socket callback and custom pointer */
  curl_multi_setopt(multi, CURLMOPT_NOTIFYFUNCTION, notify_cb);
  curl_multi_setopt(multi, CURLMOPT_NOTIFYDATA, &setup);
  curl_multi_notify_enable(multi, CURLMNOTIFY_INFO_READ);
}
```

AVAILABILITY

Added in curl 8.17.0

RETURN VALUE

Returns CURLM_OK.

SEE ALSO

CURLMOPT_NOTIFYDATA(3), **curl_multi_notify_disable**(3), **curl_multi_notify_enable**(3), **curl_multi_socket_action**(3)