## NAME

CURLMOPT_SOCKETFUNCTION − callback informed about what to wait for

## SYNOPSIS

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

int socket_callback(CURL *easy,      /* easy handle */
                    curl_socket_t s, /* socket */
                    int what,        /* describes the socket */
                    void *clientp,   /* private callback pointer */
                    void *socketp);  /* private socket pointer */

CURLMcode curl_multi_setopt(CURLM *handle, CURLMOPT_SOCKETFUNCTION, socket_callback);
```

## DESCRIPTION

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

When the *curl_multi_socket_action(3)* function is called, it uses this callback to inform the application about updates in the socket (file descriptor) status by doing none, one, or multiple calls to the **socket_callback**. The callback function gets status updates with changes since the previous time the callback was called. If the given callback pointer is set to NULL, no callback is called.

libcurl then expects the application to monitor the sockets for the specific activities and tell libcurl again when something happens on one of them. Tell libcurl by calling *curl_multi_socket_action(3)*.

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

## CALLBACK ARGUMENTS

*easy* identifies the specific transfer for which this update is related. Since this callback manages a whole multi handle, an application should not make assumptions about which particular handle that is passed here. It might even be an internal easy handle that the application did not add itself.

*s* is the specific socket this function invocation concerns. If the **what** argument is not CURL_POLL_REMOVE then it holds information about what activity on this socket the application is supposed to monitor. Subsequent calls to this callback might update the **what** bits for a socket that is already monitored.

The socket callback should return 0 on success, and −1 on error. If this callback returns error, **all** transfers currently in progress in this multi handle are aborted and made to fail.

**clientp** is set with *CURLMOPT_SOCKETDATA(3)*.

**socketp** is set with *curl_multi_assign(3)* or NULL.

The **what** parameter informs the callback on the status of the given socket. It can hold one of these values:

CURL_POLL_IN
: Wait for incoming data. For the socket to become readable.

CURL_POLL_OUT
: Wait for outgoing data. For the socket to become writable.

CURL_POLL_INOUT
: Wait for incoming and outgoing data. For the socket to become readable or writable.

CURL_POLL_REMOVE
: The specified socket/file descriptor is no longer used by libcurl for any active transfer. It might soon be added again.

## DEFAULT

NULL (no callback)

## PROTOCOLS

This functionality affects all supported protocols

## EXAMPLE

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

static int sock_cb(CURL *e, curl_socket_t s, int what, void *cbp, void *sockp)
{
  struct priv *p = cbp;
  printf("our ptr: %p\n", p->ours);

  if(what == CURL_POLL_REMOVE) {
    /* remove the socket from our collection */
  }
  if(what & CURL_POLL_IN) {
    /* wait for read on this socket */
  }
  if(what & CURL_POLL_OUT) {
    /* wait for write on this socket */
  }

  return 0;
}

int main(void)
{
  struct priv setup;
  CURLM *multi = curl_multi_init();
  /* ... use socket callback and custom pointer */
  curl_multi_setopt(multi, CURLMOPT_SOCKETFUNCTION, sock_cb);
  curl_multi_setopt(multi, CURLMOPT_SOCKETDATA, &setup);
}
```

## AVAILABILITY

Added in curl 7.15.4

## RETURN VALUE

Returns CURLM_OK.

## SEE ALSO

**CURLMOPT_SOCKETDATA**(3), **CURLMOPT_TIMERFUNCTION**(3), **curl_multi_socket_action**(3)