NAME

CURLOPT_OPENSOCKETFUNCTION − callback for opening socket

SYNOPSIS

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

typedef enum {
  CURLSOCKTYPE_IPCXN,  /* socket created for a specific IP connection */
} curlsocktype;

struct curl_sockaddr {
  int family;
  int socktype;
  int protocol;
  unsigned int addrlen;
  struct sockaddr addr;
};

curl_socket_t opensocket_callback(void *clientp,
                                  curlsocktype purpose,
                                  struct curl_sockaddr *address);

CURLcode curl_easy_setopt(CURL *handle, CURLOPT_OPENSOCKETFUNCTION, opensocket_callback);
```

DESCRIPTION

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

This callback function gets called by libcurl instead of the *socket(2)* call. The callback's *purpose* argument identifies the exact purpose for this particular socket. *CURLSOCKTYPE_IPCXN* is for IP based connections and is the only purpose currently used in libcurl. Future versions of libcurl may support more purposes.

The *clientp* pointer contains whatever user−defined value set using the *CURLOPT_OPENSOCKETDATA(3)* function.

The callback gets the resolved peer address as the *address* argument and is allowed to modify the address or refuse to connect completely. The callback function should return the newly created socket or *CURL_SOCKET_BAD* in case no connection could be established or another error was detected. Any additional *setsockopt(2)* calls can of course be done on the socket at the user's discretion.

If *CURL_SOCKET_BAD* is returned by the callback then libcurl treats it as a failed connection and tries to open a socket to connect to a different IP address associated with the transfer. If there are no more addresses to try then libcurl fails the transfer with error code *CURLE_COULDNT_CONNECT*.

You can get the IP address that curl is opening the socket for by casting *address−>addr* to *sockaddr_in* if *address−>family* is *AF_INET*, or to *sockaddr_in6* if *address−>family* is *AF_INET6*. For an example of how that data can be compared against refer to *docs/examples/block_ip.c*.

If you want to pass in a socket with an already established connection, pass the socket back with this callback and then use *CURLOPT_SOCKOPTFUNCTION(3)* to signal that it already is connected.

DEFAULT

The equivalent of this:

```
return socket(addr->family, addr->socktype, addr->protocol);
```

PROTOCOLS

This functionality affects all supported protocols

EXAMPLE

```
/* make libcurl use the already established socket 'sockfd' */

static curl_socket_t opensocket(void *clientp,
                                curlsocktype purpose,
                                struct curl_sockaddr *address)
{
  curl_socket_t sockfd;
  sockfd = *(curl_socket_t *)clientp;
  /* the actual externally set socket is passed in via the OPENSOCKETDATA
     option */
  return sockfd;
}

static int sockopt_callback(void *clientp, curl_socket_t curlfd,
                            curlsocktype purpose)
{
  return CURL_SOCKOPT_ALREADY_CONNECTED;
}

int main(void)
{
  CURL *curl = curl_easy_init();
  if(curl) {
    CURLcode result;
    extern int sockfd; /* the already connected one */
    /* libcurl thinks that you connect to the host
     * and port that you specify in the URL option. */
    curl_easy_setopt(curl, CURLOPT_URL, "http://99.99.99.99:9999");
    /* call this function to get a socket */
    curl_easy_setopt(curl, CURLOPT_OPENSOCKETFUNCTION, opensocket);
    curl_easy_setopt(curl, CURLOPT_OPENSOCKETDATA, &sockfd);

    /* call this function to set options for the socket */
    curl_easy_setopt(curl, CURLOPT_SOCKOPTFUNCTION, sockopt_callback);

    result = curl_easy_perform(curl);

    curl_easy_cleanup(curl);
  }
}
```

AVAILABILITY

Added in curl 7.17.1

RETURN VALUE

curl_easy_setopt(3) returns a CURLcode indicating success or error.

CURLE_OK (0) means everything was OK, non−zero means an error occurred, see *libcurl−errors(3)*.

SEE ALSO

CURLOPT_CLOSESOCKETFUNCTION(3), **CURLOPT_OPENSOCKETFUNCTION**(3), **CURLOPT_SOCKOPTFUNCTION**(3)