NAME

curl_easy_recv - receives raw data on an "easy" connection

SYNOPSIS

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

CURLcode curl_easy_recv(CURL *curl, void *buffer, size_t buflen, size_t *n);
```

DESCRIPTION

This function receives raw data from the established connection. You may use it together with *curl_easy_send(3)* to implement custom protocols using libcurl. This functionality can be particularly useful if you use proxies and/or SSL encryption: libcurl takes care of proxy negotiation and connection setup.

buffer is a pointer to your buffer memory that gets populated by the received data. **buflen** is the maximum amount of data you can get in that buffer. The variable **n** points to receives the number of received bytes.

To establish the connection, set *CURLOPT_CONNECT_ONLY(3)* option before calling *curl_easy_perform(3)* or *curl_multi_perform(3)*. Note that *curl_easy_recv(3)* does not work on connections that were created without this option.

The call returns **CURLE_AGAIN** if there is no data to read - the socket is used in non-blocking mode internally. When **CURLE_AGAIN** is returned, use your operating system facilities like *select(2)* to wait for data. The socket may be obtained using *curl_easy_getinfo(3)* with *CURLINFO_ACTIVESOCKET(3)*.

Wait on the socket only if *curl_easy_recv(3)* returns **CURLE_AGAIN**. The reason for this is libcurl or the SSL library may internally cache some data, therefore you should call *curl_easy_recv(3)* until all data is read which would include any cached data.

Furthermore if you wait on the socket and it tells you there is data to read, *curl_easy_recv(3)* may return **CURLE_AGAIN** if the only data that was read was for internal SSL processing, and no other data is available.

EXAMPLE

```
int main(void)
{
  CURL *curl = curl_easy_init();
  if(curl) {
    CURLcode res;
    curl_easy_setopt(curl, CURLOPT_URL, "https://example.com");
    /* Do not do the transfer - only connect to host */
    curl_easy_setopt(curl, CURLOPT_CONNECT_ONLY, 1L);
    res = curl_easy_perform(curl);

    if(res == CURLE_OK) {
      char buf[256];
      size_t nread;
      long sockfd;

      /* Extract the socket from the curl handle - we need it for waiting. */
      res = curl_easy_getinfo(curl, CURLINFO_ACTIVESOCKET, &sockfd);

      /* read data */
      res = curl_easy_recv(curl, buf, sizeof(buf), &nread);
    }
  }
}
```

AVAILABILITY

Added in 7.18.2.

RETURN VALUE

On success, returns **CURLE_OK**, stores the received data into **buffer**, and the number of bytes it actually read into ***n**.

On failure, returns the appropriate error code.

The function may return **CURLE_AGAIN**. In this case, use your operating system facilities to wait until data can be read, and retry.

Reading exactly 0 bytes indicates a closed connection.

If there is no socket available to use from the previous transfer, this function returns **CURLE_UNSUPPORTED_PROTOCOL**.

SEE ALSO

curl_easy_getinfo(3), **curl_easy_perform**(3), **curl_easy_send**(3), **curl_easy_setopt**(3)