NAME

CURLOPT_HEADERFUNCTION − callback that receives header data

SYNOPSIS

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

size_t header_callback(char *buffer,
                       size_t size,
                       size_t nitems,
                       void *userdata);

CURLcode curl_easy_setopt(CURL *handle, CURLOPT_HEADERFUNCTION, header_callback);
```

DESCRIPTION

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

This function gets called by libcurl as soon as it has received header data. The header callback will be called once for each header and only complete header lines are passed on to the callback. Parsing headers is very easy using this. The size of the data pointed to by *buffer* is *size* multiplied with *nmemb*. Do not assume that the header line is zero terminated! The pointer named *userdata* is the one you set with the *CURLOPT_HEADERDATA(3)* option. This callback function must return the number of bytes actually taken care of. If that amount differs from the amount passed in to your function, it'll signal an error to the library. This will cause the transfer to get aborted and the libcurl function in progress will return *CURLE_WRITE_ERROR*.

A complete HTTP header that is passed to this function can be up to *CURL_MAX_HTTP_HEADER* (100K) bytes.

If this option is not set, or if it is set to NULL, but *CURLOPT_HEADERDATA(3)* is set to anything but NULL, the function used to accept response data will be used instead. That is, it will be the function specified with *CURLOPT_WRITEFUNCTION(3)*, or if it is not specified or NULL - the default, stream-writing function.

It's important to note that the callback will be invoked for the headers of all responses received after initiating a request and not just the final response. This includes all responses which occur during authentication negotiation. If you need to operate on only the headers from the final response, you will need to collect headers in the callback yourself and use HTTP status lines, for example, to delimit response boundaries.

When a server sends a chunked encoded transfer, it may contain a trailer. That trailer is identical to a HTTP header and if such a trailer is received it is passed to the application using this callback as well. There are several ways to detect it being a trailer and not an ordinary header: 1) it comes after the response-body. 2) it comes after the final header line (CR LF) 3) a Trailer: header among the regular response-headers mention what header(s) to expect in the trailer.

For non-HTTP protocols like FTP, POP3, IMAP and SMTP this function will get called with the server responses to the commands that libcurl sends.

DEFAULT

Nothing.

PROTOCOLS

Used for all protocols with headers or meta-data concept: HTTP, FTP, POP3, IMAP, SMTP and more.

EXAMPLE

```
static size_t header_callback(char *buffer, size_t size,
                              size_t nitems, void *userdata)
{
```

```
 /* received header is nitems * size long in 'buffer' NOT ZERO TERMINATED */
 /* 'userdata' is set with CURLOPT_WRITEDATA */
 return nitems * size;
}

CURL *curl = curl_easy_init();
if(curl) {
 curl_easy_setopt(curl, CURLOPT_URL, "http://example.com");

 curl_easy_setopt(curl, CURLOPT_HEADERFUNCTION, header_callback);

 curl_easy_perform(curl);
}
```

AVAILABILITY

Always

RETURN VALUE

Returns CURLE_OK

SEE ALSO

CURLOPT_HEADERDATA(3), **CURLOPT_WRITEFUNCTION**(3),