NAME

CURLOPT_HEADERDATA − pointer to pass to header callback

SYNOPSIS

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

CURLcode curl_easy_setopt(CURL *handle, CURLOPT_HEADERDATA, void *pointer);
```

DESCRIPTION

Pass a *pointer* to be used to write the header part of the received data to.

If *CURLOPT_WRITEFUNCTION(3)* or *CURLOPT_HEADERFUNCTION(3)* is used, *pointer* is passed in to the respective callback.

If neither of those options are set, *pointer* must be a valid FILE * and it is used by a plain fwrite() to write headers to.

If you are using libcurl as a Windows DLL, you **must** use a *CURLOPT_WRITEFUNCTION(3)* or *CURLOPT_HEADERFUNCTION(3)* if you set this option or you might experience crashes.

DEFAULT

NULL

PROTOCOLS

This functionality affects all supported protocols

EXAMPLE

```
struct my_info {
  int shoesize;
  char *secret;
};

static size_t header_callback(char *buffer, size_t size,
                              size_t nitems, void *userdata)
{
  struct my_info *i = userdata;
  printf("shoe size: %d\n", i->shoesize);
  /* now this callback can access the my_info struct */

  return nitems * size;
}

int main(void)
{
  CURL *curl = curl_easy_init();
  if(curl) {
    CURLcode result;
    struct my_info my = { 10, "the cookies are in the cupboard" };
    curl_easy_setopt(curl, CURLOPT_URL, "https://example.com");

    curl_easy_setopt(curl, CURLOPT_HEADERFUNCTION, header_callback);

    /* pass in custom data to the callback */
    curl_easy_setopt(curl, CURLOPT_HEADERDATA, &my);

    result = curl_easy_perform(curl);
```

```
    curl_easy_cleanup(curl);
  }
}
```

AVAILABILITY

Added in curl 7.10

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_HEADERFUNCTION(3), **CURLOPT_WRITEFUNCTION**(3), **curl_easy_header**(3)