NAME

CURLOPT_DEBUGFUNCTION − debug callback

SYNOPSIS

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

typedef enum {
  CURLINFO_TEXT = 0,
  CURLINFO_HEADER_IN,    /* 1 */
  CURLINFO_HEADER_OUT,   /* 2 */
  CURLINFO_DATA_IN,      /* 3 */
  CURLINFO_DATA_OUT,     /* 4 */
  CURLINFO_SSL_DATA_IN,  /* 5 */
  CURLINFO_SSL_DATA_OUT, /* 6 */
  CURLINFO_END
} curl_infotype;

int debug_callback(CURL *handle,
                   curl_infotype type,
                   char *data,
                   size_t size,
                   void *clientp);

CURLcode curl_easy_setopt(CURL *handle, CURLOPT_DEBUGFUNCTION,
                          debug_callback);
```

DESCRIPTION

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

CURLOPT_DEBUGFUNCTION(3) replaces the standard debug function used when *CURLOPT_VERBOSE(3)* is in effect. This callback receives debug information, as specified in the *type* argument. This function must return 0. The *data* pointed to by the char * passed to this function is not null−terminated, but is exactly of the *size* as told by the *size* argument.

WARNING: this callback may receive sensitive contents from headers and data, including information sent as **CURLINFO_TEXT**.

The *clientp* argument is the pointer set with *CURLOPT_DEBUGDATA(3)*.

Available **curl_infotype** values:

CURLINFO_TEXT
: The data is informational text.

CURLINFO_HEADER_IN
: The data is header (or header−like) data received from the peer.

CURLINFO_HEADER_OUT
: The data is header (or header−like) data sent to the peer.

CURLINFO_DATA_IN
: The data is the unprocessed protocol data received from the peer. Even if the data is encoded or compressed, it is not provided decoded nor decompressed to this callback. If you need the data in decoded and decompressed form, use *CURLOPT_WRITEFUNCTION(3)*.

CURLINFO_DATA_OUT
: The data is protocol data sent to the peer.

CURLINFO_SSL_DATA_OUT
: The data is SSL/TLS (binary) data sent to the peer.

CURLINFO_SSL_DATA_IN
: The data is SSL/TLS (binary) data received from the peer.

WARNING: This callback may be called with the curl *handle* set to an internal handle. (Added in 8.4.0)

If you need to distinguish your curl *handle* from internal handles then set *CURLOPT_PRIVATE(3)* on your handle.

DEFAULT

NULL

PROTOCOLS

This functionality affects all supported protocols

EXAMPLE

```
static void dump(const char *text,
                 FILE *stream, unsigned char *ptr, size_t size)
{
  size_t i;
  size_t c;
  unsigned int width = 0x10;

  fprintf(stream, "%s, %lu bytes (0x%lx)\n",
          text, (long)size, (long)size);

  for(i = 0; i < size; i += width) {
    fprintf(stream, "%4.4lx: ", (long)i);

    /* show hex to the left */
    for(c = 0; c < width; c++) {
      if(i + c < size)
        fprintf(stream, "%02x ", ptr[i + c]);
      else
        fputs("   ", stream);
    }

    /* show data on the right */
    for(c = 0; (c < width) && (i + c < size); c++) {
      char x = (ptr[i + c] >= 0x20 && ptr[i + c] < 0x80) ? ptr[i + c] : '.';
      fputc(x, stream);
    }

    fputc('\n', stream); /* newline */
  }
}

static int my_trace(CURL *handle, curl_infotype type,
                    char *data, size_t size,
                    void *clientp)
{
  const char *text;
  (void)handle;
  (void)clientp;
```

```
  switch(type) {
  case CURLINFO_TEXT:
    fputs("== Info: ", stderr);
    fwrite(data, size, 1, stderr);
    return 0;
  default: /* in case a new one is introduced to shock us */
    return 0;

  case CURLINFO_HEADER_OUT:
    text = "=> Send header";
    break;
  case CURLINFO_DATA_OUT:
    text = "=> Send data";
    break;
  case CURLINFO_SSL_DATA_OUT:
    text = "=> Send SSL data";
    break;
  case CURLINFO_HEADER_IN:
    text = "<= Recv header";
    break;
  case CURLINFO_DATA_IN:
    text = "<= Recv data";
    break;
  case CURLINFO_SSL_DATA_IN:
    text = "<= Recv SSL data";
    break;
  }

  dump(text, stderr, (unsigned char *)data, size);
  return 0;
}

int main(void)
{
  CURL *curl;
  CURLcode result;

  curl = curl_easy_init();
  if(curl) {
    curl_easy_setopt(curl, CURLOPT_DEBUGFUNCTION, my_trace);

    /* the DEBUGFUNCTION has no effect until we enable VERBOSE */
    curl_easy_setopt(curl, CURLOPT_VERBOSE, 1L);

    /* example.com is redirected, so we tell libcurl to follow redirection */
    curl_easy_setopt(curl, CURLOPT_FOLLOWLOCATION, 1L);

    curl_easy_setopt(curl, CURLOPT_URL, "https://example.com/");
    result = curl_easy_perform(curl);
    /* Check for errors */
    if(result != CURLE_OK)
      fprintf(stderr, "curl_easy_perform() failed: %s\n",
              curl_easy_strerror(result));
```

```
  /* always cleanup */
  curl_easy_cleanup(curl);
 }
 return 0;
}
```

AVAILABILITY

Added in curl 7.9.6

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

CURLINFO_CONN_ID(3), **CURLINFO_XFER_ID**(3), **CURLOPT_DEBUGDATA**(3), **CURLOPT_VERBOSE**(3), **curl_global_trace**(3)