NAME

curl_mime_data_cb - set a callback-based data source for a mime part’s body

SYNOPSIS

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

size_t readfunc(char *buffer, size_t size, size_t nitems, void *arg);

int seekfunc(void *arg, curl_off_t offset, int origin);

void freefunc(void *arg);

CURLcode curl_mime_data_cb(curl_mimepart *part, curl_off_t datasize,
                           curl_read_callback readfunc,
                           curl_seek_callback seekfunc,
                           curl_free_callback freefunc, void *arg);
```

DESCRIPTION

curl_mime_data_cb(3) sets the data source of a mime part’s body content from a data read callback function.

part is the part’s to assign contents to.

readfunc is a pointer to a data read callback function, with a signature as shown by the above prototype. It may not be set to NULL.

seekfunc is a pointer to a seek callback function, with a signature as shown by the above prototype. This function is used when resending data (i.e.: after a redirect); this pointer may be set to NULL, in which case a resend might not be not possible.

freefunc is a pointer to a user resource freeing callback function, with a signature as shown by the above prototype. If no resource is to be freed, it may safely be set to NULL. This function is called upon mime structure freeing.

arg is a user defined argument to callback functions.

The read callback function gets called by libcurl as soon as it needs to read data in order to send it to the peer - like if you ask it to upload or post data to the server. The data area pointed at by the pointer *buffer* should be filled up with at most *size* multiplied with *nitems* number of bytes by your function.

Your read function must then return the actual number of bytes that it stored in that memory area. Returning 0 signals end-of-file to the library and cause it to stop the current transfer.

If you stop the current transfer by returning 0 "pre-maturely" (i.e. before the server expected it, like when you have said you intend to upload N bytes and yet you upload less than N bytes), you may experience that the server "hangs" waiting for the rest of the data that does not come.

The read callback may return *CURL_READFUNC_ABORT* to stop the current operation immediately, resulting in a *CURLE_ABORTED_BY_CALLBACK* error code from the transfer.

The callback can return *CURL_READFUNC_PAUSE* to cause reading from this connection to pause. See *curl_easy_pause(3)* for further details.

The seek function gets called by libcurl to rewind input stream data or to seek to a certain position. The function shall work like fseek(3) or lseek(3) and it gets SEEK_SET, SEEK_CUR or SEEK_END as argument for *origin*, although libcurl currently only passes SEEK_SET.

The callback function must return *CURL_SEEKFUNC_OK* on success, *CURL_SEEKFUNC_FAIL* to cause the upload operation to fail or *CURL_SEEKFUNC_CANTSEEK* to indicate that while the seek failed, libcurl is free to work around the problem if possible. The latter can sometimes be done by instead reading from the input or similar.

Care must be taken if the part is bound to a curl easy handle that is later duplicated: the *arg* pointer argument is also duplicated, resulting in the pointed item to be shared between the original and the copied handle. In particular, special attention should be given to the *freefunc* procedure code since it then gets called twice with the same argument.

PROTOCOLS

This functionality affects http, imap and smtp

EXAMPLE

Sending a huge data string causes the same amount of memory to be allocated: to avoid overhead resources consumption, one might want to use a callback source to avoid data duplication. In this case, original data must be retained until after the transfer terminates.

```
#include <string.h> /* for memcpy */
char hugedata[512000];

struct ctl {
  char *buffer;
  curl_off_t size;
```

```
  curl_off_t position;
};

size_t read_callback(char *buffer, size_t size, size_t nitems, void *arg)
{
  struct ctl *p = (struct ctl *) arg;
  curl_off_t sz = p->size - p->position;

  nitems *= size;
  if(sz > nitems)
    sz = nitems;
  if(sz)
    memcpy(buffer, p->buffer + p->position, sz);
  p->position += sz;
  return sz;
}

int seek_callback(void *arg, curl_off_t offset, int origin)
{
  struct ctl *p = (struct ctl *) arg;

  switch(origin) {
  case SEEK_END:
    offset += p->size;
    break;
  case SEEK_CUR:
    offset += p->position;
    break;
  }

  if(offset < 0)
    return CURL_SEEKFUNC_FAIL;
  p->position = offset;
  return CURL_SEEKFUNC_OK;
}

int main(void)
{
  CURL *curl = curl_easy_init();
  if(curl) {
```

```
    curl_mime *mime = curl_mime_init(curl);
    curl_mimepart *part = curl_mime_addpart(mime);
    struct ctl hugectl;

    hugectl.buffer = hugedata;
    hugectl.size = sizeof(hugedata);
    hugectl.position = 0;
    curl_mime_data_cb(part, hugectl.size, read_callback, seek_callback, NULL,
                      &hugectl);
  }
}
```

AVAILABILITY

Added in curl 7.56.0

RETURN VALUE

CURLE_OK or a CURL error code upon failure.

SEE ALSO

curl_easy_duphandle(3), **curl_mime_addpart**(3), **curl_mime_data**(3), **curl_mime_name**(3)