NAME

curl_ws_start_frame − start a new WebSocket frame

SYNOPSIS

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

CURLcode curl_ws_start_frame(CURL *curl,
                             unsigned int flags,
                             curl_off_t frame_len);
```

DESCRIPTION

Add the WebSocket frame header for the given flags and length to the transfers send buffer for WebSocket encoded data. Intended for use in a *CURLOPT_READFUNCTION(3)* callback.

When using a *CURLOPT_READFUNCTION(3)* in a WebSocket transfer, any data returned by that function is sent as a *CURLWS_BINARY* frame with the length being the amount of data read.

To send larger frames or frames of a different type, call curl_ws_start_frame() from within the read function and then return the data belonging to the frame.

The function fails, if a previous frame has not been completely read yet. Also it fails in *CURLWS_RAW_MODE*.

The read function in libcurl usually treats a return value of 0 as the end of file indication and stops any further reads. This would prevent sending WebSocket frames of length 0.

If the read function calls *curl_ws_start_frame()*, a return value of 0 is *not* treated as an end of file and libcurl calls the read function again.

FLAGS

Supports all flags documented in *curl_ws_meta(3)*.

PROTOCOLS

This functionality affects ws only

EXAMPLE

```
#include <string.h> /* for strlen */

struct read_ctx {
  CURL *easy;
  char *message;
  size_t msg_len;
  size_t nsent;
};

static size_t readcb(char *buf, size_t nitems, size_t buflen, void *p)
{
  struct read_ctx *ctx = p;
  size_t len = nitems * buflen;
  size_t left = ctx->msg_len - ctx->nsent;

  if(!ctx->nsent) {
    CURLcode result;
    /* Want to send TEXT frame. */
    result = curl_ws_start_frame(ctx->easy, CURLWS_TEXT,
                                 (curl_off_t)ctx->msg_len);
    if(result != CURLE_OK) {
```

```
      fprintf(stderr, "error starting frame: %d\n", result);
      return CURL_READFUNC_ABORT;
    }
  }
  if(left) {
    if(left < len)
      len = left;
    memcpy(buf, ctx->message + ctx->nsent, len);
    ctx->nsent += len;
    return len;
  }
  return 0;
}

int main(void)
{
  CURL *easy;
  struct read_ctx rctx;
  CURLcode result;

  easy = curl_easy_init();
  if(!easy)
    return 1;

  curl_easy_setopt(easy, CURLOPT_URL, "wss://example.com");
  curl_easy_setopt(easy, CURLOPT_READFUNCTION, readcb);
  /* tell curl that we want to send the payload */
  memset(&rctx, 0, sizeof(rctx));
  rctx.easy = easy;
  rctx.message = "Hello, friend!";
  rctx.msg_len = strlen(rctx.message);
  curl_easy_setopt(easy, CURLOPT_READDATA, &rctx);
  curl_easy_setopt(easy, CURLOPT_UPLOAD, 1L);

  /* Perform the request, result gets the return code */
  result = curl_easy_perform(easy);
  /* Check for errors */
  if(result != CURLE_OK)
    fprintf(stderr, "curl_easy_perform() failed: %s\n",
            curl_easy_strerror(result));

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

AVAILABILITY

Added in curl 8.16.0

RETURN VALUE

This function returns a CURLcode indicating success or error.

CURLE_OK (0) means everything was OK, non−zero means an error occurred, see *libcurl−errors(3)*. If *CURLOPT_ERRORBUFFER(3)* was set with *curl_easy_setopt(3)* there can be an error message stored in the error buffer when non−zero is returned.

Instead of blocking, the function returns **CURLE_AGAIN**. The correct behavior is then to wait for the socket to signal readability before calling this function again.

Any other non−zero return value indicates an error. See the *libcurl−errors(3)* man page for the full list with descriptions.

SEE ALSO

curl_easy_getinfo(3), **curl_easy_perform**(3), **curl_easy_setopt**(3), **curl_ws_recv**(3), **libcurl-ws**(3)