## NAME

curl_formadd - add a section to a multipart form POST

## SYNOPSIS

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

CURLFORMcode curl_formadd(struct curl_httppost **firstitem,
                          struct curl_httppost **lastitem, ...);
```

## DESCRIPTION

**This function is deprecated.** Use *curl_mime_init(3)* instead.

curl_formadd() is used to append sections when building a multipart form post. Append one section at a time until you have added all the sections you want included and then you pass the *firstitem* pointer as parameter to *CURLOPT_HTTPPOST(3)*. *lastitem* is set after each *curl_formadd(3)* call and on repeated invokes it should be left as set to allow repeated invokes to find the end of the list faster.

After the *lastitem* pointer follow the real arguments.

The pointers *firstitem* and *lastitem* should both be pointing to NULL in the first call to this function. All list-data is allocated by the function itself. You must call *curl_formfree(3)* on the *firstitem* after the form post has been done to free the resources.

Using POST with HTTP 1.1 implies the use of a "Expect: 100-continue" header. You can disable this header with *CURLOPT_HTTPHEADER(3)* as usual.

First, there are some basics you need to understand about multipart form posts. Each part consists of at least a NAME and a CONTENTS part. If the part is made for file upload, there are also a stored CONTENT-TYPE and a FILENAME. Below, we discuss what options you use to set these properties in the parts you want to add to your post.

The options listed first are for making normal parts. The options from *CURLFORM_FILE* through *CURLFORM_BUFFERLENGTH* are for file upload parts.

## OPTIONS

CURLFORM_COPYNAME

followed by a string which provides the *name* of this part. libcurl copies the string so your application does not need to keep it around after this function call. If the name is not null-terminated, you must set its length with **CURLFORM_NAMELENGTH**. The *name* is not allowed to contain zero-valued bytes. The copied data is freed by *curl_formfree(3)*.

CURLFORM_PTRNAME
: followed by a string which provides the *name* of this part. libcurl uses the pointer and refer to the data in your application, so you must make sure it remains until curl no longer needs it. If the name is not null-terminated, you must set its length with **CURLFORM_NAMELENGTH**. The *name* is not allowed to contain zero-valued bytes.

CURLFORM_COPYCONTENTS
: followed by a pointer to the contents of this part, the actual data to send away. libcurl copies the provided data, so your application does not need to keep it around after this function call. If the data is not null terminated, or if you would like it to contain zero bytes, you must set the length of the name with **CURLFORM_CONTENTSLENGTH**. The copied data is freed by *curl_formfree(3)*.

CURLFORM_PTRCONTENTS
: followed by a pointer to the contents of this part, the actual data to send away. libcurl uses the pointer and refer to the data in your application, so you must make sure it remains until curl no longer needs it. If the data is not null-terminated, or if you would like it to contain zero bytes, you must set its length with **CURLFORM_CONTENTSLENGTH**.

CURLFORM_CONTENTLEN
: followed by a curl_off_t value giving the length of the contents. Note that for *CURLFORM_STREAM* contents, this option is mandatory.

  If you pass a 0 (zero) for this option, libcurl calls strlen() on the contents to figure out the size. If you really want to send a zero byte content then you must make sure strlen() on the data pointer returns zero.

  (Option added in 7.46.0)

CURLFORM_CONTENTSLENGTH
: (This option is deprecated. Use *CURLFORM_CONTENTLEN* instead!)

  followed by a long giving the length of the contents. Note that for *CURLFORM_STREAM* contents, this option is mandatory.

  If you pass a 0 (zero) for this option, libcurl calls strlen() on the contents to figure out the size. If you really want to send a zero byte content then you must make sure strlen() on the data pointer returns zero.

CURLFORM_FILECONTENT
: followed by a filename, causes that file to be read and its contents used as data in this part. This

part does *not* automatically become a file upload part simply because its data was read from a file.

The specified file needs to kept around until the associated transfer is done.

CURLFORM_FILE
: followed by a filename, makes this part a file upload part. It sets the *filename* field to the basename of the provided filename, it reads the contents of the file and passes them as data and sets the content-type if the given file match one of the internally known file extensions. For **CURLFORM_FILE** the user may send one or more files in one part by providing multiple **CURLFORM_FILE** arguments each followed by the filename (and each *CURLFORM_FILE* is allowed to have a *CURLFORM_CONTENTTYPE*).

The given upload file has to exist in its full in the file system already when the upload starts, as libcurl needs to read the correct file size beforehand.

The specified file needs to kept around until the associated transfer is done.

CURLFORM_CONTENTTYPE
: is used in combination with *CURLFORM_FILE*. Followed by a pointer to a string which provides the content-type for this part, possibly instead of an internally chosen one.

CURLFORM_FILENAME
: is used in combination with *CURLFORM_FILE*. Followed by a pointer to a string, it tells libcurl to use the given string as the *filename* in the file upload part instead of the actual file name.

CURLFORM_BUFFER
: is used for custom file upload parts without use of *CURLFORM_FILE*. It tells libcurl that the file contents are already present in a buffer. The parameter is a string which provides the *filename* field in the content header.

CURLFORM_BUFFERPTR
: is used in combination with *CURLFORM_BUFFER*. The parameter is a pointer to the buffer to be uploaded. This buffer must not be freed until after *curl_easy_cleanup(3)* is called. You must also use *CURLFORM_BUFFERLENGTH* to set the number of bytes in the buffer.

CURLFORM_BUFFERLENGTH
: is used in combination with *CURLFORM_BUFFER*. The parameter is a long which gives the length of the buffer.

CURLFORM_STREAM

Tells libcurl to use the *CURLOPT_READFUNCTION(3)* callback to get data. The parameter you pass to *CURLFORM_STREAM* is the pointer passed on to the read callback's fourth argument. If you want the part to look like a file upload one, set the *CURLFORM_FILENAME* parameter as well. Note that when using *CURLFORM_STREAM*, *CURLFORM_CONTENTSLENGTH* must also be set with the total expected length of the part unless the formpost is sent chunked encoded. (Option added in libcurl 7.18.2)

CURLFORM_ARRAY

Another possibility to send options to curl_formadd() is the **CURLFORM_ARRAY** option, that passes a struct curl_forms array pointer as its value. Each curl_forms structure element has a *CURLformoption* and a char pointer. The final element in the array must be a CURLFORM_END. All available options can be used in an array, except the CURLFORM_ARRAY option itself. The last argument in such an array must always be **CURLFORM_END**.

CURLFORM_CONTENTHEADER

specifies extra headers for the form POST section. This takes a curl_slist prepared in the usual way using **curl_slist_append** and appends the list of headers to those libcurl automatically generates. The list must exist while the POST occurs, if you free it before the post completes you may experience problems.

When you have passed the *struct curl_httppost* pointer to *curl_easy_setopt(3)* (using the *CURLOPT_HTTPPOST(3)* option), you must not free the list until after you have called *curl_easy_cleanup(3)* for the curl handle.

See example below.

## EXAMPLE

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

static const char record[]="data in a buffer";

int main(void)
{
  CURL *curl = curl_easy_init();
  if(curl) {
    struct curl_httppost *post = NULL;
    struct curl_httppost *last = NULL;
    char namebuffer[] = "name buffer";
    long namelength = strlen(namebuffer);
    char buffer[] = "test buffer";
```

```
char htmlbuffer[] = "<HTML>test buffer</HTML>";
long htmlbufferlength = strlen(htmlbuffer);
struct curl_forms forms[3];
char file1[] = "my-face.jpg";
char file2[] = "your-face.jpg";
/* add null character into htmlbuffer, to demonstrate that
   transfers of buffers containing null characters actually work
*/
htmlbuffer[8] = '\0';

/* Add simple name/content section */
curl_formadd(&post, &last, CURLFORM_COPYNAME, "name",
        CURLFORM_COPYCONTENTS, "content", CURLFORM_END);

/* Add simple name/content/contenttype section */
curl_formadd(&post, &last, CURLFORM_COPYNAME, "htmlcode",
        CURLFORM_COPYCONTENTS, "<HTML></HTML>",
        CURLFORM_CONTENTTYPE, "text/html", CURLFORM_END);

/* Add name/ptrcontent section */
curl_formadd(&post, &last, CURLFORM_COPYNAME, "name_for_ptrcontent",
        CURLFORM_PTRCONTENTS, buffer, CURLFORM_END);

/* Add ptrname/ptrcontent section */
curl_formadd(&post, &last, CURLFORM_PTRNAME, namebuffer,
        CURLFORM_PTRCONTENTS, buffer, CURLFORM_NAMELENGTH,
        namelength, CURLFORM_END);

/* Add name/ptrcontent/contenttype section */
curl_formadd(&post, &last, CURLFORM_COPYNAME, "html_code_with_hole",
        CURLFORM_PTRCONTENTS, htmlbuffer,
        CURLFORM_CONTENTSLENGTH, htmlbufferlength,
        CURLFORM_CONTENTTYPE, "text/html", CURLFORM_END);

/* Add simple file section */
curl_formadd(&post, &last, CURLFORM_COPYNAME, "picture",
        CURLFORM_FILE, "my-face.jpg", CURLFORM_END);

/* Add file/contenttype section */
curl_formadd(&post, &last, CURLFORM_COPYNAME, "picture",
```

```
            CURLFORM_FILE, "my-face.jpg",
            CURLFORM_CONTENTTYPE, "image/jpeg", CURLFORM_END);

    /* Add two file section */
    curl_formadd(&post, &last, CURLFORM_COPYNAME, "pictures",
            CURLFORM_FILE, "my-face.jpg",
            CURLFORM_FILE, "your-face.jpg", CURLFORM_END);

    /* Add two file section using CURLFORM_ARRAY */
    forms[0].option = CURLFORM_FILE;
    forms[0].value  = file1;
    forms[1].option = CURLFORM_FILE;
    forms[1].value  = file2;
    forms[2].option  = CURLFORM_END;

    /* Add a buffer to upload */
    curl_formadd(&post, &last,
            CURLFORM_COPYNAME, "name",
            CURLFORM_BUFFER, "data",
            CURLFORM_BUFFERPTR, record,
            CURLFORM_BUFFERLENGTH, sizeof(record),
            CURLFORM_END);

    /* no option needed for the end marker */
    curl_formadd(&post, &last, CURLFORM_COPYNAME, "pictures",
            CURLFORM_ARRAY, forms, CURLFORM_END);
    /* Add the content of a file as a normal post text value */
    curl_formadd(&post, &last, CURLFORM_COPYNAME, "filecontent",
            CURLFORM_FILECONTENT, ".bashrc", CURLFORM_END);
    /* Set the form info */
    curl_easy_setopt(curl, CURLOPT_HTTPPOST, post);

    curl_easy_perform(curl);

    curl_easy_cleanup(curl);

    curl_formfree(post);
  }
}
```

## AVAILABILITY

Deprecated in 7.56.0. Before this release, field names were allowed to contain zero-valued bytes. The pseudo-filename "-" to read stdin is discouraged although still supported, but data is not read before being actually sent: the effective data size can then not be automatically determined, resulting in a chunked encoding transfer. Backslashes and double quotes in field and file names are now escaped before transmission.

## RETURN VALUE

0 means everything was OK, non-zero means an error occurred corresponding to a CURL_FORMADD_* constant defined in <*curl/curl.h*>

## SEE ALSO

**curl_easy_setopt**(3), **curl_formfree**(3), **curl_mime_init**(3)