

1. **http.**

**2. Include definitions.**

```
⟨include files 2⟩ ≡  
#include "platform.h"  
#include <microhttpd.h>  
#include <assert.h>
```

This code is used in sections 11 and 25.

**3. declarations.**

⟨declarations of functions 3⟩ ≡

```
int cb_request(void *cls, struct MHD_Connection *connection, const char *url, const char
    *method, const char *version, const char *upload_data, size_t *upload_data_size, void **ptr);
void logger(void *cls, const char *fm, va_list ap);
```

This code is used in sections 11 and 25.

**4.**

⟨type declarations 4⟩ ≡

```
typedef struct _Request *Request;
```

This code is used in section 11.

**5.**

⟨accept policy callback option 5⟩ ≡

```
Λ, Λ ,
```

This code is used in section 11.

**6.**

⟨http request callback option 6⟩ ≡

```
&cb_request, Λ ,
```

This code is used in section 11.

**7.**

⟨http options 7⟩ ≡

```
MHD_OPTION_CONNECTION_TIMEOUT, 256 ,
```

This code is used in section 11.

**8.** Define HTTPS related options. The key and a certificate needs to be set.

⟨https specific options 8⟩ ≡

```
MHD_OPTION_HTTPS_MEM_KEY, key_pem, MHD_OPTION_HTTPS_MEM_CERT, cert_pem ,
```

**9.** ⟨logging options 9⟩ ≡

```
MHD_OPTION_EXTERNAL_LOGGER, logger, &argv ,
```

**10.** ⟨library functions 10⟩ ≡

```
void logger(void *cls, const char *fm, va_list ap)
{
    fprintf(stderr, "!!!!");
    vfprintf(stderr, fm, ap);
    fprintf(stderr, "\n");
}
```

See also sections 15, 16, 18, 21, and 22.

This code is used in section 25.

**11. main.**

```

<include files 2>
<declarations of functions 3>
<type declarations 4>
<local functions 24>
int main(int argc, char *const *argv)
{
    struct MHD_Daemon *d;
    assert(MHD_is_feature_supported(MHD_FEATURE_MESSAGES));
    if (argc ≠ 2) {
        printf("%s_PORT\n", argv[0]);
        return 1;
    }
    unsigned int flags = MHD_USE_THREAD_PER_CONNECTION;
    flags |= MHD_USE_INTERNAL_POLLING_THREAD;
    flags |= MHD_USE_ERROR_LOG;
    d = MHD_start_daemon(flags,
        atoi(argv[1]),
        <accept policy callback option 5>
        <http request callback option 6>
        <http options 7>
        MHD_OPTION_END);
    if (d ≡ Λ) return 1;
    (void) getc(stdin);
    MHD_stop_daemon(d);
    return 0;
}

```

**12. processing.****13.**

```

⟨try open file 13⟩ ≡
    FILE *file = fopen(&url[1], "rb");
    struct stat buf;
    if (Λ ≠ file) {
        int fd = fileno(file);
        if (-1 ≡ fd) {
            fclose(file);
            file = Λ;
        }
        else if ((0 ≠ fstat(fd, &buf)) ∨ (¬S_ISREG(buf.st_mode))) {
            /* not a regular file, refuse to serve */
            fclose(file);
            file = Λ;
        }
    }
}

```

This code is used in section 22.

**14. respond with data in file by using callbacks for data and for cleanup.**

```

⟨respond page from file content 14⟩ ≡
    status_code = MHD_HTTP_OK;
    response = MHD_create_response_from_callback(buf.st_size, 32 * 1024, /* 32k size */
        &file_reader, file, &file_free_callback);

```

This code is used in section 22.

**15. file callback**

```

⟨library functions 10⟩ +≡
    static ssize_t file_reader(void *cls, uint64_t pos, char *buf, size_t max)
    {
        FILE *file = cls;
        (void) fseek(file, pos, SEEK_SET);
        return fread(buf, 1, max, file);
    }

```

**16. file cleanup callback**

```

⟨library functions 10⟩ +≡
    static void file_free_callback(void *cls)
    {
        fclose((FILE *) cls);
    }

```

**17.**

```

⟨respond static page 17⟩ ≡
    response = MHD_create_response_from_buffer(strlen(page), (void *) page, MHD_RESPMEM_PERSISTENT);

```

This code is used in section 22.

18.  $\langle$  library functions 10  $\rangle + \equiv$

```
int print_key_value(void *cls, enum MHD_ValueKind kind, const char *key, const char *value)
{
    fprintf(stderr, "***_d:%s:%s\n", kind, key, value);
    return MHD_YES;
}
```

19.  $\langle$  check for allowed method 19  $\rangle \equiv$

```
if ((0  $\neq$  strcmp(method, MHD_HTTP_METHOD_GET))  $\wedge$  (0  $\neq$  strcmp(method, MHD_HTTP_METHOD_HEAD)))
    return MHD_NO; /* unexpected method */
```

20.  $\langle$  log request info 20  $\rangle \equiv$

```
fprintf(stderr, "ECHO_url:%s\n_method:%s\n", url, method);
fprintf(stderr, "upload_data_size:%d\n", *upload_data_size);
MHD_get_connection_values(connection,
    MHD_HEADER_KIND | MHD_COOKIE_KIND | MHD_POSTDATA_KIND | MHD_FOOTER_KIND, print_key_value,  $\Lambda$ );
```

This code is used in section 22.

21.  $\langle$  library functions 10  $\rangle + \equiv$

```
enum MHD_Result post_iterator(void *cls, enum MHD_ValueKind kind, const char *key, const
    char *filename, const char *content_type, const char *transfer_encoding, const char
    *data, uint64_t off, size_t size)
{
    struct Request *request = cls;
    fprintf(stderr, "###_s\n", key);
    return MHD_YES;
}
```

22.  $\langle$ library functions 10 $\rangle + \equiv$

```
enum MHD_Result cb_request(void *cls, struct MHD_Connection *connection, const char *url, const
    char *method, const char *version, const char *upload_data, size_t *upload_data_size, void
    **ptr)
{
    static char *page = "{ \"data\": 1 }";
    static int aptr;
    struct MHD_Response *response = Λ;
    int ret;
    unsigned int status_code = MHD_HTTP_NOT_IMPLEMENTED;
     $\langle$ log request info 20 $\rangle$ 
    if ((0  $\equiv$  strcmp(method, MHD_HTTP_METHOD_POST))) {
        fprintf(stderr, "Upload_data_size: %d\n", *upload_data_size);
        if (*upload_data_size  $\equiv$  0) return MHD_YES;
        else {
            fprintf(stderr, "CONTENT: ");
            for (int i = 0; i < *upload_data_size; i++) {
                fprintf(stderr, "%02x", upload_data[i]);
            }
            for (int i = 0; i < *upload_data_size; i++) {
                fprintf(stderr, "%c", upload_data[i]);
            }
            fprintf(stderr, "%p\n", response);
            response = MHD_create_response_from_buffer(*upload_data_size, upload_data,
                MHD_RESPMEM_MUST_COPY);
            MHD_add_response_header(response, MHD_HTTP_HEADER_CONTENT_ENCODING, "application/json");
            *upload_data_size = 0;
            status_code = MHD_HTTP_OK;
        }
    }
    if (response  $\equiv$  Λ) {
         $\langle$ try open file 13 $\rangle$ 
        if (file) {
             $\langle$ respond page from file content 14 $\rangle$ 
        }
    }
    if (response  $\equiv$  Λ) {
         $\langle$ respond static page 17 $\rangle$ 
    }
    fprintf(stderr, "response %p\n", response);
    ret = MHD_queue_response(connection, status_code, response);
    fprintf(stderr, "queued_response %d -> %d\n", status_code, ret);
    MHD_destroy_response(response);
    return ret;
}
```

23.  $\langle$ library helper functions 23 $\rangle \equiv$  /\* empty \*/

This code is used in section 25.

24.  $\langle$ local functions 24 $\rangle \equiv$  /\* empty \*/

This code is used in section 11.

**25.** library

```
⟨dummy.c 25⟩ ≡  
  ⟨include files 2⟩  
  ⟨declarations of functions 3⟩  
  ⟨library helper functions 23⟩  
  ⟨library functions 10⟩
```

**26.**

```
⟨initialize request local data 26⟩ ≡  
  if (&aptr ≠ *ptr) { /* do never respond on first call */  
    *ptr = &aptr;  
    return MHD_YES;  
  }
```



**27. INDEX.**

*\_Request*: 4.  
*ap*: 3, 10.  
*aptr*: 22, 26.  
*argc*: 11.  
*argv*: 9, 11.  
*assert*: 11.  
*atoi*: 11.  
*buf*: 13, 14, 15.  
*cb\_request*: 3, 6, 22.  
*cert\_pem*: 8.  
*cls*: 3, 10, 15, 16, 18, 21, 22.  
*connection*: 3, 20, 22.  
*content\_type*: 21.  
*d*: 11.  
*data*: 21.  
*fclose*: 13, 16.  
*fd*: 13.  
*file*: 13, 14, 15, 22.  
*file\_free\_callback*: 14, 16.  
*file\_reader*: 14, 15.  
*filename*: 21.  
*fileno*: 13.  
*flags*: 11.  
*fm*: 3, 10.  
*fopen*: 13.  
*fprintf*: 10, 18, 20, 21, 22.  
*fread*: 15.  
*fseek*: 15.  
*fstat*: 13.  
*getc*: 11.  
*i*: 22.  
*key*: 18, 21.  
*key\_pem*: 8.  
*kind*: 18, 21.  
*logger*: 3, 9, 10.  
*main*: 11.  
*max*: 15.  
*method*: 3, 19, 20, 22.  
*MHD\_add\_response\_header*: 22.  
*MHD\_Connection*: 3, 22.  
*MHD\_COOKIE\_KIND*: 20.  
*MHD\_create\_response\_from\_buffer*: 17, 22.  
*MHD\_create\_response\_from\_callback*: 14.  
*MHD\_Daemon*: 11.  
*MHD\_destroy\_response*: 22.  
*MHD\_FEATURE\_MESSAGES*: 11.  
*MHD\_FOOTER\_KIND*: 20.  
*MHD\_get\_connection\_values*: 20.  
*MHD\_HEADER\_KIND*: 20.  
*MHD\_HTTP\_HEADER\_CONTENT\_ENCODING*: 22.  
*MHD\_HTTP\_METHOD\_GET*: 19.  
*MHD\_HTTP\_METHOD\_HEAD*: 19.  
*MHD\_HTTP\_METHOD\_POST*: 22.  
*MHD\_HTTP\_NOT\_IMPLEMENTED*: 22.  
*MHD\_HTTP\_OK*: 14, 22.  
*MHD\_is\_feature\_supported*: 11.  
*MHD\_NO*: 19.  
*MHD\_OPTION\_CONNECTION\_TIMEOUT*: 7.  
*MHD\_OPTION\_END*: 11.  
*MHD\_OPTION\_EXTERNAL\_LOGGER*: 9.  
*MHD\_OPTION\_HTTPS\_MEM\_CERT*: 8.  
*MHD\_OPTION\_HTTPS\_MEM\_KEY*: 8.  
*MHD\_POSTDATA\_KIND*: 20.  
*MHD\_queue\_response*: 22.  
*MHD\_RESPMEM\_MUST\_COPY*: 22.  
*MHD\_RESPMEM\_PERSISTENT*: 17.  
*MHD\_Response*: 22.  
*MHD\_Result*: 21, 22.  
*MHD\_start\_daemon*: 11.  
*MHD\_stop\_daemon*: 11.  
*MHD\_USE\_ERROR\_LOG*: 11.  
*MHD\_USE\_INTERNAL\_POLLING\_THREAD*: 11.  
*MHD\_USE\_THREAD\_PER\_CONNECTION*: 11.  
*MHD\_ValueKind*: 18, 21.  
*MHD\_YES*: 18, 21, 22, 26.  
*off*: 21.  
*page*: 17, 22.  
*pos*: 15.  
*post\_iterator*: 21.  
*print\_key\_value*: 18, 20.  
*printf*: 11.  
*ptr*: 3, 22, 26.  
**Request**: 4, 21.  
*request*: 21.  
*response*: 14, 17, 22.  
*ret*: 22.  
*S\_ISREG*: 13.  
*SEEK\_SET*: 15.  
*size*: 21.  
*ssize\_t*: 15.  
*st\_mode*: 13.  
*st\_size*: 14.  
*stat*: 13.  
*status\_code*: 14, 22.  
*stderr*: 10, 18, 20, 21, 22.  
*stdin*: 11.  
*strcmp*: 19, 22.  
*strlen*: 17.  
*transfer\_encoding*: 21.  
*uint64\_t*: 15, 21.  
*upload\_data*: 3, 22.  
*upload\_data\_size*: 3, 20, 22.

*url*: [3](#), [13](#), [20](#), [22](#).

*value*: [18](#).

*version*: [3](#), [22](#).

*vfprintf*: [10](#).

⟨accept policy callback option 5⟩ Used in section 11.  
⟨check for allowed method 19⟩  
⟨declarations of functions 3⟩ Used in sections 11 and 25.  
⟨dummy.c 25⟩  
⟨http options 7⟩ Used in section 11.  
⟨http request callback option 6⟩ Used in section 11.  
⟨https specific options 8⟩  
⟨include files 2⟩ Used in sections 11 and 25.  
⟨initialize request local data 26⟩  
⟨library functions 10, 15, 16, 18, 21, 22⟩ Used in section 25.  
⟨library helper functions 23⟩ Used in section 25.  
⟨local functions 24⟩ Used in section 11.  
⟨log request info 20⟩ Used in section 22.  
⟨logging options 9⟩  
⟨respond page from file content 14⟩ Used in section 22.  
⟨respond static page 17⟩ Used in section 22.  
⟨try open file 13⟩ Used in section 22.  
⟨type declarations 4⟩ Used in section 11.

# HTTP

	Section	Page
http .....	<a href="#">1</a>	1
Include defintions .....	<a href="#">2</a>	2
declarations .....	<a href="#">3</a>	3
main .....	<a href="#">11</a>	4
processing .....	<a href="#">12</a>	5
INDEX .....	<a href="#">27</a>	9