

1. **http.**

2. File Structure.**3. Main Program.**

```

<include files 5>
<declarations of functions 7>
<type declarations 8>
<local functions 27>
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 9>
        <http request callback option 10>
        <http options 11>
        <logging options 13>
        MHD_OPTION_END);
    if (d ≡ Λ) return 1;
    (void) getc(stdin);
    MHD_stop_daemon(d);
    return 0;
}

```

4. library

```

<dummy.c 4> ≡
<include files 5>
<declarations of functions 7>
<library helper functions 26>
<library functions 14>

```

```

5. <include files 5> ≡
#include "platform.h"
#include <microhttpd.h>
#include <assert.h>
#include <stdbool.h>

```

This code is used in sections 3 and 4.

```

6. <initialize request local data 6> ≡
if (&aptr ≠ *ptr) { /* do never respond on first call */
    *ptr = &aptr;
    return MHD_YES;
}

```

7. declarations.

⟨declarations of functions 7⟩ ≡

```
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);
void logger(void *cls, const char *fm, va_list ap);
```

This code is used in sections 3 and 4.

8.

⟨type declarations 8⟩ ≡

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

This code is used in section 3.

9.

⟨accept policy callback option 9⟩ ≡

```
Λ, Λ ,
```

This code is used in section 3.

10.

⟨http request callback option 10⟩ ≡

```
&cb_request, Λ ,
```

This code is used in section 3.

11.

⟨http options 11⟩ ≡

```
MHD_OPTION_CONNECTION_TIMEOUT, 256 ,
```

This code is used in section 3.

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

⟨https specific options 12⟩ ≡

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

13. ⟨logging options 13⟩ ≡

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

This code is used in section 3.

14. ⟨library functions 14⟩ ≡

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

See also sections 18, 19, 21, 24, and 25.

This code is used in section 4.

15. processing.**16.**

```

⟨try open file 16⟩ ≡
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 25.

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

```

⟨respond page from file content 17⟩ ≡
    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 25.

18. file callback

```

⟨library functions 14⟩ +=
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);
}

```

19. file cleanup callback

```

⟨library functions 14⟩ +=
static void file_free_callback(void *cls)
{
    fclose((FILE *) cls);
}

```

20.

```

⟨respond static page 20⟩ ≡
    response = MHD_create_response_from_buffer(strlen(page), (void *) page, MHD_RESPMEM_PERSISTENT);

```

This code is used in section 25.

21. \langle library functions 14 $\rangle + \equiv$

```
enum MHD_Result 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;
}
```

22. \langle check for allowed method 22 $\rangle \equiv$

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

23. \langle log request info 23 $\rangle \equiv$

```
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 25.

24. \langle library functions 14 $\rangle + \equiv$

```
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;
}
```

25. \langle library functions 14 $\rangle + \equiv$

```
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 23 $\rangle$ 
    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);
            const char *xpage = "XXX";
            if (false) response = MHD_create_response_from_buffer(*upload_data_size, (void *)
                upload_data, MHD_RESPMEM_MUST_COPY);
            else response = MHD_create_response_from_buffer(strlen(xpage), (void *) xpage,
                MHD_RESPMEM_MUST_COPY);
            MHD_add_response_header(response, MHD_HTTP_HEADER_CONTENT_ENCODING, "application/json");
            if (false) *upload_data_size = 0;
            status_code = MHD_HTTP_OK;
            ret = MHD_queue_response(connection, status_code, response);
            fprintf(stderr, "x_queued_response %d -> %d\n", status_code, ret);
            MHD_destroy_response(response);
            return MHD_YES;
        }
    }
    if (response  $\equiv$  Λ) {
         $\langle$ try open file 16 $\rangle$ 
        if (file) {
             $\langle$ respond page from file content 17 $\rangle$ 
        }
    }
    if (response  $\equiv$  Λ) {
         $\langle$ respond static page 20 $\rangle$ 
    }
    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;
}
```

```
}
```

26. \langle library helper functions 26 $\rangle \equiv$ `/* empty */`

This code is used in section 4.

27. \langle local functions 27 $\rangle \equiv$ `/* empty */`

This code is used in section 3.

28. INDEX.

_Request: 8.
ap: 7, 14.
aptr: 6, 25.
argc: 3.
argv: 3, 13.
assert: 3.
atoi: 3.
buf: 16, 17, 18.
cb_request: 7, 10, 25.
cert_pem: 12.
cls: 7, 14, 18, 19, 21, 24, 25.
connection: 7, 23, 25.
content_type: 24.
d: 3.
data: 24.
false: 25.
fclose: 16, 19.
fd: 16.
file: 16, 17, 18, 25.
file_free_callback: 17, 19.
file_reader: 17, 18.
filename: 24.
fileno: 16.
flags: 3.
fn: 7, 14.
fopen: 16.
fprintf: 14, 21, 23, 24, 25.
fread: 18.
fseek: 18.
fstat: 16.
getc: 3.
i: 25.
key: 21, 24.
key_pem: 12.
kind: 21, 24.
logger: 7, 13, 14.
main: 3.
max: 18.
method: 7, 22, 23, 25.
MHD_add_response_header: 25.
MHD_Connection: 7, 25.
MHD_COOKIE_KIND: 23.
MHD_create_response_from_buffer: 20, 25.
MHD_create_response_from_callback: 17.
MHD_Daemon: 3.
MHD_destroy_response: 25.
MHD_FEATURE_MESSAGES: 3.
MHD_FOOTER_KIND: 23.
MHD_get_connection_values: 23.
MHD_HEADER_KIND: 23.
MHD_HTTP_HEADER_CONTENT_ENCODING: 25.
MHD_HTTP_METHOD_GET: 22.
MHD_HTTP_METHOD_HEAD: 22.
MHD_HTTP_METHOD_POST: 25.
MHD_HTTP_NOT_IMPLEMENTED: 25.
MHD_HTTP_OK: 17, 25.
MHD_is_feature_supported: 3.
MHD_NO: 22.
MHD_OPTION_CONNECTION_TIMEOUT: 11.
MHD_OPTION_END: 3.
MHD_OPTION_EXTERNAL_LOGGER: 13.
MHD_OPTION_HTTPS_MEM_CERT: 12.
MHD_OPTION_HTTPS_MEM_KEY: 12.
MHD_POSTDATA_KIND: 23.
MHD_queue_response: 25.
MHD_RESPMEM_MUST_COPY: 25.
MHD_RESPMEM_PERSISTENT: 20.
MHD_Response: 25.
MHD_Result: 7, 21, 24, 25.
MHD_start_daemon: 3.
MHD_stop_daemon: 3.
MHD_USE_ERROR_LOG: 3.
MHD_USE_INTERNAL_POLLING_THREAD: 3.
MHD_USE_THREAD_PER_CONNECTION: 3.
MHD_ValueKind: 21, 24.
MHD_YES: 6, 21, 24, 25.
off: 24.
page: 20, 25.
pos: 18.
post_iterator: 24.
print_key_value: 21, 23.
printf: 3.
ptr: 6, 7, 25.
Request: 8, 24.
request: 24.
response: 17, 20, 25.
ret: 25.
S_ISREG: 16.
SEEK_SET: 18.
size: 24.
ssize_t: 18.
st_mode: 16.
st_size: 17.
stat: 16.
status_code: 17, 25.
stderr: 14, 21, 23, 24, 25.
stdin: 3.
strcmp: 22, 25.
strlen: 20, 25.
transfer_encoding: 24.
uint64_t: 18, 24.
upload_data: 7, 25.

upload_data_size: [7](#), [23](#), [25](#).

url: [7](#), [16](#), [23](#), [25](#).

value: [21](#).

version: [7](#), [25](#).

vfprintf: [14](#).

xpage: [25](#).

⟨accept policy callback option 9⟩ Used in section 3.
⟨check for allowed method 22⟩
⟨declarations of functions 7⟩ Used in sections 3 and 4.
⟨dummy.c 4⟩
⟨http options 11⟩ Used in section 3.
⟨http request callback option 10⟩ Used in section 3.
⟨https specific options 12⟩
⟨include files 5⟩ Used in sections 3 and 4.
⟨initialize request local data 6⟩
⟨library functions 14, 18, 19, 21, 24, 25⟩ Used in section 4.
⟨library helper functions 26⟩ Used in section 4.
⟨local functions 27⟩ Used in section 3.
⟨log request info 23⟩ Used in section 25.
⟨logging options 13⟩ Used in section 3.
⟨respond page from file content 17⟩ Used in section 25.
⟨respond static page 20⟩ Used in section 25.
⟨try open file 16⟩ Used in section 25.
⟨type declarations 8⟩ Used in section 3.

HTTP

	Section	Page
http	1	1
File Structure	2	2
declarations	7	3
processing	15	4
INDEX	28	8