

KLone Reference Manual

Generated by Doxygen 1.3.4

Fri Dec 5 13:42:15 2008

Contents

1	KLone	1
1.1	Essential variables	1
1.2	Essential functions	1
2	KLone Module Index	3
2.1	KLone Modules	3
3	KLone File Index	5
3.1	KLone File List	5
4	KLone Module Documentation	9
4.1	Dynamic page interfaces	9
4.2	Request Handling	10
4.3	Response Handling	20
4.4	Sessions	27
4.5	Input/Output	29
4.6	Basic Functions	30
4.7	Filters for Compression and Encryption	38
4.8	Miscellaneous Utility Functions	40
5	KLone File Documentation	51
5.1	http.h File Reference	51
5.2	response.h File Reference	53

Chapter 1

KLone

Below is a quick reference list of essential variables and functions used by all KLone applications. A more complete list is available following the links above.

1.1 Essential variables

Any KLone dynamic page (*.kl1) can access the following predefined objects:

- **request_t**(p. 10) *request;

Used to access all information related to the current HTTP request such as form variables, cookies, HTTP header fields, uploaded files, etc.

- **response_t**(p. 20) *response;

This object lets you access and modify the HTTP response entities such as header fields and status codes.

- **session_t**(p. 27) *session;

Sessions are objects which can be used to preserve client related information throughout subsequent accesses. Simple functions are provided to set/get {name, value} pairs.

- **io_t**(p. 30) *out;

The output object is used to print out dynamically generated content.

1.2 Essential functions

- `const char* request_get_arg (request_t *rq, const char *name)`

Return the value of a form variable.

- `const char* request_get_field_value(request_t *rq, const char *name);`

Return the value of an header field.

- `int response_redirect(response_t *rs, const char *url)`

Redirect to the specified URL.

- `const char *session_get(session_t *ss, const char *name)`

Get the value of a session variable.

- `int session_set(session_t *ss, const char *name, const char *value)`

Set the value of a session variable.

Chapter 2

KLone Module Index

2.1 KLone Modules

Here is a list of all modules:

Dynamic page interfaces	9
Request Handling	10
Response Handling	20
Sessions	27
Input/Output	29
Basic Functions	30
Filters for Compression and Encryption	38
Miscellaneous Utility Functions	40

Chapter 3

KLone File Index

3.1 KLone File List

Here is a list of all documented files with brief descriptions:

<code>access.c</code>	??
<code>access.h</code>	??
<code>addr.c</code>	??
<code>addr.h</code>	??
<code>atom.c</code>	??
<code>atom.h</code>	??
<code>backend.c</code>	??
<code>backend.h</code>	??
<code>broker.c</code>	??
<code>broker.h</code>	??
<code>caes.h</code>	??
<code>ccipher.h</code>	??
<code>cgi.c</code>	??
<code>cgi.h</code>	??
<code>cgzip.h</code>	??
<code>child.c</code>	??
<code>child.h</code>	??
<code>child_nfy.c</code>	??
<code>cipher.c</code>	??
<code>cnull.h</code>	??
<code>codec.c</code>	??
<code>codec.h</code>	??
<code>codecs.h</code>	??
<code>common.c</code>	??
<code>context.h</code>	??
<code>date.c</code>	??
<code>dypage.c</code>	??
<code>dypage.h</code>	??
<code>emb.c</code>	??
<code>emb.h</code>	??
<code>embpage.h</code>	??
<code>entry.c</code>	??
<code>field.c</code>	??

field.h	??
file.c	??
gzip.c	??
header.c	??
header.h	??
hook.c	??
hook.h	??
hookprv.h	??
http.c	??
http.h	51
http_s.h	??
io.c	??
io.h	??
iofd.c	??
iomem.c	??
iopriv.h	??
iossl.c	??
kilt.h	??
kilt_urls.h	??
klog.c	??
klog.h	??
klogprv.h	??
klone.h	??
kloned/main.c	??
tools/klone/main.c	??
main.h	??
md5.c	??
md5.h	??
mem.c	??
mime_map.c	??
mime_map.h	??
null.c	??
os.h	??
page.h	??
parser.c	??
parser.h	??
path.c	??
pm.c	??
pm.h	??
ppc.c	??
ppc.h	??
ppc_access_log.c	??
ppc_cmd.h	??
ppc_fork_child.c	??
ppc_log_add.c	??
ppc_log_get.c	??
ppc_nop.c	??
pwd.c	??
req.c	??
request.c	??
request.h	??
response.c	??
response.h	53
rsfilter.c	??

rsfilter.h	??
run.h	??
server.c	??
server.h	??
server_nfy.c	??
server_ppc_cmd.h	??
server_s.h	??
ses_client.c	??
ses_file.c	??
ses_mem.c	??
ses_prv.h	??
session.c	??
session.h	??
sup_cgi.c	??
sup_emb.c	??
sup_fs.c	??
sup_kilt.c	??
supplier.h	??
syslog.c	??
timer.c	??
timer.h	??
tls.c	??
tls.h	??
tls_dh_autogen.c	??
tls_glue.c	??
tls_psk.c	??
tlsprv.h	??
trans_c.c	??
translat.c	??
translat.h	??
uc.h	??
utils.c	??
utils.h	??
va.h	??
var.c	??
var.h	??
varprv.h	??
vars.c	??
vars.h	??
version.c	??
version.h	??
vhost.c	??
vhost.h	??

Chapter 4

KLone Module Documentation

4.1 Dynamic page interfaces

Modules

- **Request Handling**
- **Response Handling**
- **Sessions**

4.1.1 Detailed Description

Basic knowledge of the HTTP protocol is assumed. Hence only the essential information is given. Some useful references are:

- RFC 2616 for a complete description of HTTP 1.1 header fields
- RFC 2109 for cookie format
- RFC 822 for standard data type formats
- <http://www.iana.org/assignments/media-types/> for an updated list of possible mime-types

4.2 Request Handling

Functions

- `io_t * request_io (request_t *rq)`
Get the io_t object associated with a request object.
- `vars_t * request_get_cookies (request_t *rq)`
Get the cookies list.
- `const char * request_get_cookie (request_t *rq, const char *name)`
Get the value of a cookie named name.
- `vars_t * request_get_args (request_t *rq)`
Get the list of request variables (using either GET or POST method).
- `vars_t * request_get_getargs (request_t *rq)`
Get request variables passed using the GET method.
- `vars_t * request_get_postargs (request_t *rq)`
Get request variables passed using the POST method.
- `const char * request_get_arg (request_t *rq, const char *name)`
Get a request variable (using either GET or POST variable list).
- `const char * request_get_getarg (request_t *rq, const char *name)`
Return a GET variable.
- `const char * request_get_postarg (request_t *rq, const char *name)`
Return a POST variable.
- `const char * request_get_uri (request_t *rq)`
Get the URI field of a request.
- `const char * request_get_filename (request_t *rq)`
Get the filename field of a request.
- `const char * request_get_query_string (request_t *rq)`
Get the query string field of a request.
- `const char * request_get_path_info (request_t *rq)`
Get the path info field of a request.
- `time_t request_get_if_modified_since (request_t *rq)`
Get IMS field of a request.
- `http_t * request_get_http (request_t *rq)`
Get the HTTP server handle of a request.
- `const char * request_get_client_request (request_t *rq)`

Return the client request line.

- `ssize_t request_get_content_length` (`request_t *rq`)
Get the content length of a request.
- `vars_t * request_get_uploads` (`request_t *rq`)
Get uploaded files.
- `int request_get_uploaded_file` (`request_t *rq`, `const char *name`, `size_t idx`, `char local_filename[U_FILENAME_MAX]`, `char client_filename[U_FILENAME_MAX]`, `char mime_type[MIME_TYPE_BUFSZ]`, `size_t *file_size`)
Get info and handles of an uploaded file.
- `int request_get_method` (`request_t *rq`)
Get the method of a request.
- `const char * request_get_protocol` (`request_t *rq`)
Get the protocol used by the client request.
- `const char * request_get_resolved_filename` (`request_t *rq`)
Get resolved filename of a request.
- `const char * request_get_resolved_path_info` (`request_t *rq`)
Get the resolved path info of a request.
- `kaddr_t * request_get_addr` (`request_t *rq`)
Return the local address.
- `kaddr_t * request_get_peer_addr` (`request_t *rq`)
Return the peer address.
- `header_t * request_get_header` (`request_t *rq`)
Return the header object.
- `field_t * request_get_field` (`request_t *rq`, `const char *name`)
Get an header field.
- `const char * request_get_field_value` (`request_t *rq`, `const char *name`)
Get the value of an header field.

4.2.1 Function Documentation

4.2.1.1 `io_t* request_io (request_t * rq)`

Return the I/O object (`io_t`) used by the request object passed as parameter. The `io_t` object is bound to the socket connected to the client (the web browser).

Parameters:

`rq` request object

Returns:

child `io_t` object of the given `rq` or NULL if no `io_t` object has been set

See also:

`io_t`

Definition at line 140 of file `request.c`.

4.2.1.2 vars_t* request_get_cookies (request_t * rq)

Return a `vars_t` object containing the list of all cookies sent by the browser.

Parameters:

rq request object

Returns:

the cookie list of the given `rq`

Definition at line 158 of file `request.c`.

4.2.1.3 const char* request_get_cookie (request_t * rq, const char * name)

Return the value of a cookie sent by the browser

Parameters:

rq request object

name cookie name

Returns:

the cookie value or NULL on error

Definition at line 176 of file `request.c`.

4.2.1.4 vars_t* request_get_args (request_t * rq)

Return get/post arguments of request `rq` in a `vars_t` object

Parameters:

rq request object

Returns:

the arguments' list of the given `rq`

Definition at line 198 of file `request.c`.

4.2.1.5 vars_t* request_get_getargs (request_t * rq)

Return GET arguments of request `rq` in a `vars_t` object

Parameters:

rq request object

Returns:

the arguments' list of the given *rq*

Definition at line 215 of file request.c.

4.2.1.6 vars_t* request_get_postargs (request_t * *rq*)

Return POST arguments of request *rq* in a *vars_t* object

Parameters:

rq request object

Returns:

the arguments' list of the given *rq*

Definition at line 232 of file request.c.

4.2.1.7 const char* request_get_arg (request_t * *rq*, const char * *name*)

Return the string value of argument *name* in request *rq*.

Parameters:

rq request object

name name of the argument

Returns:

- the string value corresponding to the supplied *name*
- NULL if there's no argument named *name*

Definition at line 252 of file request.c.

4.2.1.8 const char* request_get_getarg (request_t * *rq*, const char * *name*)

Return the string value of argument *name* in request *rq*.

Parameters:

rq request object

name name of the argument

Returns:

- the string value corresponding to the supplied *name*
- NULL if there's no argument named *name*

Definition at line 277 of file request.c.

4.2.1.9 `const char* request_get_postarg (request_t * rq, const char * name)`

Return the string value of argument *name* in request *rq*.

Parameters:

- rq* request object
- name* name of the argument

Returns:

- the string value corresponding to the supplied *name*
- NULL if there's no argument named *name*

Definition at line 302 of file request.c.

4.2.1.10 `const char* request_get_uri (request_t * rq)`

Return the string value of the URI in request *rq*.

Parameters:

- rq* request object

Returns:

- the URI child object of the given *rq*

Definition at line 333 of file request.c.

4.2.1.11 `const char* request_get_filename (request_t * rq)`

Return the string value of the filename field in request *rq*.

Parameters:

- rq* request object

Returns:

- the file name bound to *rq* (can be NULL)

Definition at line 350 of file request.c.

4.2.1.12 `const char* request_get_query_string (request_t * rq)`

Return the query string field of request *rq*.

Parameters:

- rq* request object

Returns:

- the query string bound to *rq* (may be NULL)

Definition at line 390 of file request.c.

4.2.1.13 `const char* request_get_path_info (request_t * rq)`

Return the path info field of request `rq`.

Parameters:

`rq` request object

Returns:

the path info of `rq` (may be `NULL`)

Definition at line 407 of file `request.c`.

4.2.1.14 `time_t request_get_if_modified_since (request_t * rq)`

Return the `time_t` value of the IMS field of request `rq`

Parameters:

`rq` request object

Returns:

a valid `time_t` value on success, -1 on failure

Definition at line 441 of file `request.c`.

4.2.1.15 `http_t* request_get_http (request_t * rq)`

Get the `http_t` object containing the HTTP server handle of request `rq`

Parameters:

`rq` request object

Returns:

the child HTTP object of the given `rq` (may be `NULL`)

Definition at line 481 of file `request.c`.

4.2.1.16 `const char* request_get_client_request (request_t * rq)`

Return the client request (METHOD URI HTTP/HTTP_VERSION)

Parameters:

`rq` request object

Returns:

0 if successful, non-zero on error

Definition at line 721 of file `request.c`.

4.2.1.17 `ssize_t request_get_content_length (request_t * rq)`

Retrieve a `size_t` corresponding to the *Content-Length* field of request `rq`

Parameters:

rq request object

Returns:

the content length of the given `rq`

Definition at line 908 of file `request.c`.

4.2.1.18 `vars_t* request_get_uploads (request_t * rq)`

Return the list of uploaded files.

Any `var_t` in the list will contain, within its name/value pair, the name of the HTML form input tag "name" argument and the filename (with full path) of the temporary file where uploaded content has been stored.

This function is only useful to enumerate uploads,

See also:

`request_get_uploaded_file`(p.16) is what you'll probably use.

Parameters:

rq request object

Returns:

the arguments' list of the given `rq`

Definition at line 1223 of file `request.c`.

4.2.1.19 `int request_get_uploaded_file (request_t * rq, const char * name, size_t idx, char local_filename[U_FILENAME_MAX], char client_filename[U_FILENAME_MAX], char mime_type[MIME_TYPE_BUFSZ], size_t * file_size)`

Return information and handles about an uploaded file

Parameters:

rq request object

name form input tag variable name (<input type=file name="xxx">)

idx if more then one file with the same name param exists *idx* will be used as an index (use 0 otherwise)

local_filename on successfull exit will get the filename (with full path) of the temporary file where uploaded file content has been saved. Must be at least `U_FILENAME_MAX` bytes long

client_filename filename as provided by the client Must be at least `U_FILENAME_MAX` bytes long

mime_type MIME type as stated y the client (may be "") Must be at least `MIME_TYPE_BUFSZ` bytes long

file_size file size of the uploaded file

Returns:

0 on success, ~0 otherwise

Definition at line 1335 of file request.c.

References request_get_uploaded_file().

Referenced by request_get_uploaded_file().

4.2.1.20 int request_get_method (request_t * rq)

Return the method of request *rq*. Refer to **http.h**(p. 51) for possible methods.

Parameters:

rq request object

Returns:

- the method of the given *rq*

Definition at line 1728 of file request.c.

References HM_UNKNOWN.

4.2.1.21 const char* request_get_protocol (request_t * rq)

Return the protocol of request *rq* ("HTTP/1.0", "HTTP/1.1", etc.)

Parameters:

rq request object

Returns:

- the method of the given *rq*

Definition at line 1746 of file request.c.

4.2.1.22 const char* request_get_resolved_filename (request_t * rq)

Return a string representing the resolved filename of request *rq*.

Parameters:

rq request object

Returns:

the resolved file name bound to *rq* (may be NULL)

Definition at line 1763 of file request.c.

4.2.1.23 `const char* request_get_resolved_path_info (request_t * rq)`

Return a string representing the resolved path info of request `rq`.

Parameters:

rq request object

Returns:

the resolved path info of the given `rq` (may be NULL)

Definition at line 1780 of file `request.c`.

4.2.1.24 `kaddr_t* request_get_addr (request_t * rq)`

Return the IP address and port of the server end of the socket

Parameters:

rq request object

Returns:

a pointer to an `kaddr_t` type

Definition at line 1963 of file `request.c`.

4.2.1.25 `kaddr_t* request_get_peer_addr (request_t * rq)`

Return the IP address and port of the client connected to the web server

Parameters:

rq request object

Returns:

a pointer to an `kaddr_t` type

Definition at line 1980 of file `request.c`.

4.2.1.26 `header_t* request_get_header (request_t * rq)`

Return the header object

Parameters:

rq request object

Returns:

a pointer to an `header_t` type

See also:

`header_t`

Definition at line 1998 of file `request.c`.

4.2.1.27 `field_t* request_get_field (request_t * rq, const char * name)`

Return the header field named `name`.

Parameters:

rq request object

name the name of the field

Returns:

the header field or NULL if the header is not found

Definition at line 2016 of file request.c.

4.2.1.28 `const char* request_get_field_value (request_t * rq, const char * name)`

Return the value of an header field named `name`.

Parameters:

rq request object

name the name of the field

Returns:

the value of the field or NULL if the field is not found

Definition at line 2035 of file request.c.

4.3 Response Handling

Functions

- int **response_set_content_encoding** (response_t *rs, const char *encoding)
Set response content encoding field.
- int **response_disable_caching** (response_t *rs)
Add all header field that enable page caching (i.e. disable caching).
- int **response_enable_caching** (response_t *rs)
Remove all headers that inhibit page caching.
- int **response_set_cookie** (response_t *rs, const char *name, const char *value, time_t expire, const char *path, const char *domain, int secure)
Set the value of a cookie.
- void **response_set_method** (response_t *rs, int method)
Set the response method.
- int **response_get_method** (response_t *rs)
Get the response method.
- int **response_print_header** (response_t *rs)
Print a response header.
- int **response_set_field** (response_t *rs, const char *name, const char *value)
Set an header field of a response object.
- int **response_del_field** (response_t *rs, const char *name)
Remove an header field of a response object.
- int **response_set_content_type** (response_t *rs, const char *mime_type)
Set the content type of a response to a mime type.
- int **response_set_date** (response_t *rs, time_t date)
Set the date field in a response header.
- int **response_set_last_modified** (response_t *rs, time_t mtime)
Set the last modified field in a response header.
- int **response_set_content_length** (response_t *rs, size_t sz)
Set the content length field of a response header.
- int **response_get_status** (response_t *rs)
Get the status of a response.
- header_t * **response_get_header** (response_t *rs)
Get the header of a response.

- `io_t * response_io` (`response_t *rs`)
Get the I/O object of a response.
- `int response_redirect` (`response_t *rs, const char *url`)
Redirect to a given url.
- `int response_set_status` (`response_t *rs, int status`)
Set the status of a response.

4.3.1 Function Documentation

4.3.1.1 `int response_set_content_encoding` (`response_t * rs, const char * encoding`)

Set the *Content-Encoding* field in a response object `rs` to `encoding`.

Parameters:

`rs` response object

`encoding` encoding type

Returns:

- 0 if successful
- ~0 if successful

Definition at line 44 of file response.c.

4.3.1.2 `int response_disable_caching` (`response_t * rs`)

Adds all relevant Header fields to the current HTTP response to avoid browser caching.

The function will set/modify the following fields:

Cache-Control: no-cache, must-revalidate Expires: Mon, 1 Jan 1990 05:00:00 GMT Pragma: no-cache

Parameters:

`rs` response object

Returns:

- 0 if successful
- ~0 if successful

Definition at line 74 of file response.c.

References `response_set_field()`.

4.3.1.3 int response_enable_caching (response_t * rs)

Remove all HTTP Header fields that (should) prevent browsers caching. This should enable caching on specs-compliant browsers.

Those fields are:

Cache-Control: Expires: Pragma:

Parameters:

rs response object

Returns:

- 0 if successful
- ~0 if successful

Definition at line 108 of file response.c.

References response_del_field().

4.3.1.4 int response_set_cookie (response_t * rs, const char * name, const char * value, time_t expire, const char * path, const char * domain, int secure)

Set the value of a cookie named *name* to *value* in response object *rs*. Other fields that can be set are *expire*, *path*, *domain*, and *secure*.

Parameters:

rs response object

name cookie name

value cookie value

expire cookie expiration date

path cookie path

domain cookie domain

secure cookie secure flag

Returns:

- 0 if successful
- ~0 on error

Definition at line 140 of file response.c.

References u_tt_to_rfc822(), and u_urlncpy().

4.3.1.5 void response_set_method (response_t * rs, int method)

Set the response method of *rs* to *method*. For possible values of *method*, refer to [http.h](#)(p. 51).

Parameters:

rs response object

method response method

Returns:

- nothing

Definition at line 257 of file response.c.

4.3.1.6 int response_get_method (response_t * rs)

Get the response method of **rs**. For possible values of method, refer to **http.h**(p. 51).

Parameters:

rs response object

Returns:

- the method of the given **rs**

Definition at line 274 of file response.c.

4.3.1.7 int response_print_header (response_t * rs)

Print the header of **rs**

Parameters:

rs parameter **rs** description

Returns:

- 0 if successful
- ~0 on error

Definition at line 364 of file response.c.

4.3.1.8 int response_set_field (response_t * rs, const char * name, const char * value)

Set field name to **value** in response object **rs**.

Parameters:

rs response object

name field name

value field value

Returns:

- 0 if successful
- ~0 on error

Definition at line 384 of file response.c.

Referenced by response_disable_caching().

4.3.1.9 int response_del_field (response_t * rs, const char * name)

Remove the header field whose name is **name**

Parameters:

rs response object

name field name

Returns:

- 0 if successful
- ~0 on error

Definition at line 403 of file response.c.

Referenced by response_enable_caching().

4.3.1.10 int response_set_content_type (response_t * rs, const char * mime_type)

Set the *Content-Type* field of response *rs* to *mime_type*.

Parameters:

rs response object
mime_type mime type

Returns:

- 0 if successful
- ~0 on error

Definition at line 433 of file response.c.

4.3.1.11 int response_set_date (response_t * rs, time_t date)

Set the *Date* field of *rs* to *date*.

Parameters:

rs response object
date date value

Returns:

- 0 if successful
- ~0 on error

Definition at line 457 of file response.c.

References u_tt_to_rfc822().

4.3.1.12 int response_set_last_modified (response_t * rs, time_t mtime)

Set the *Last-Modified* field of *rs* to *mtime*.

Parameters:

rs response object
mtime last modified date value

Returns:

- 0 if successful
- ~0 on error

Definition at line 484 of file response.c.

References u_tt_to_rfc822().

4.3.1.13 `int response_set_content_length (response_t * rs, size_t sz)`

Set the *Content-Length* field of `rs` to `sz`.

Parameters:

- `rs` response object
- `sz` number of bytes in content

Returns:

- 0 if successful
- ~0 on error

Definition at line 511 of file `response.c`.

4.3.1.14 `int response_get_status (response_t * rs)`

Get the status of a response `rs`. For possible values of status refer to `response.h`(p. 53).

Parameters:

- `rs` response object

Returns:

- the status of the given `rs`

Definition at line 537 of file `response.c`.

4.3.1.15 `header_t* response_get_header (response_t * rs)`

Get the header of a response `rs`.

Parameters:

- `rs` response object

Returns:

- the child header object of the given `rs`

Definition at line 553 of file `response.c`.

4.3.1.16 `io_t* response_io (response_t * rs)`

Get the I/O object of response `rs`.

Parameters:

- `rs` response object

Returns:

- the I/O child object of the given `rs`

Definition at line 569 of file `response.c`.

4.3.1.17 `int response_redirect (response_t * rs, const char * url)`

Redirect to *url* by setting the *Location* field in response *rs*.

Parameters:

- rs* parameter *rs* description
- url* parameter *url* description

Returns:

- 0 if successful
- ~0 on error

Definition at line 587 of file response.c.

References HTTP_STATUS_MOVED_TEMPORARILY, and response_set_status().

4.3.1.18 `int response_set_status (response_t * rs, int status)`

Set the *status* of response *rs*. For possible values of *status* refer to **response.h**(p. 53).

Parameters:

- rs* parameter *rs* description
- status* parameter *status* description

Returns:

- 0 always

Definition at line 616 of file response.c.

Referenced by response_redirect().

4.4 Sessions

Functions

- `vars_t * session_get_vars (session_t *ss)`
Get session variables.
- `const char * session_get (session_t *ss, const char *name)`
Get session variable with given name.
- `int session_set (session_t *ss, const char *name, const char *value)`
Set session variable with given name to a value.
- `int session_age (session_t *ss)`
Get the amount of time a session has been inactive.
- `int session_clean (session_t *ss)`
Remove all session variables.
- `int session_del (session_t *ss, const char *name)`
Delete session variable given a name.

4.4.1 Function Documentation

4.4.1.1 `vars_t* session_get_vars (session_t * ss)`

Return a `vars_t` containing the session variables.

Parameters:

ss session object

Returns:

the variables' list of the given *ss* (may be NULL)

Definition at line 524 of file session.c.

4.4.1.2 `const char* session_get (session_t * ss, const char * name)`

Return a string representation of variable in *ss* with given *name*.

Parameters:

ss session object

name session variable name

Returns:

the variable value corresponding to the given *name* (may be NULL)

Definition at line 543 of file session.c.

4.4.1.3 int session_set (session_t * ss, const char * name, const char * value)

Put variable with `name` and `value` into `ss`.

Parameters:

ss session object
name session variable name
value session variable value

Returns:

0 if successful, non-zero on error

Definition at line 566 of file session.c.

4.4.1.4 int session_age (session_t * ss)

Return the number of seconds since the session was last modified.

Parameters:

ss session object

Returns:

- the number of seconds since last modification
- -1 on error

Definition at line 606 of file session.c.

4.4.1.5 int session_clean (session_t * ss)

Remove all session variables from `ss`.

Parameters:

ss session object

Returns:

0 if successful, non-zero on error

Definition at line 628 of file session.c.

4.4.1.6 int session_del (session_t * ss, const char * name)

Delete session variable `name` in `ss`.

Parameters:

ss session object
name session variable name

Returns:

- 0 if successful
- ~0 on error

Definition at line 658 of file session.c.

4.5 Input/Output

Modules

- **Basic Functions**
- **Filters for Compression and Encryption**

4.5.1 Detailed Description

4.6 Basic Functions

Functions

- `enum io_type_e io_type (io_t *io)`
Returns the type of the given io.
- `ssize_t io_pipe (io_t *out, io_t *in)`
Write the input stream to the output stream.
- `int io_dup (io_t *io, io_t **pio)`
Duplicate an IO handle.
- `ssize_t io_copy (io_t *out, io_t *in, size_t size)`
Copy a block of data between two io_t objects.
- `ssize_t io_seek (io_t *io, size_t off)`
Seek to the given position.
- `ssize_t io_tell (io_t *io)`
Return the current file position.
- `int io_close (io_t *io)`
Close the given io object.
- `int io_free (io_t *io)`
Free an io_t object.
- `ssize_t io_read (io_t *io, char *buf, size_t size)`
Read a block of data from an io_t object.
- `ssize_t io_vprintf (io_t *io, const char *fmt, va_list ap)`
Write a string to io using printf-style va_list.
- `ssize_t io_printf (io_t *io, const char *fmt,...)`
Write a string to io using printf-style format strings.
- `ssize_t io_flush (io_t *io)`
Flush the write buffer.
- `ssize_t io_write (io_t *io, const char *buf, size_t size)`
Write a block of data to an io_t object.
- `ssize_t io_putc (io_t *io, char c)`
Write a char to an io_t object.
- `ssize_t io_getc (io_t *io, char *pc)`
Read a char from an io_t object.
- `ssize_t io_get_until (io_t *io, char stop_at, char *buf, size_t size)`

Read a chunk of data until the given character is found.

- `ssize_t io_gets (io_t *io, char *buf, size_t size)`
Read a line from an io_t object.
- `int io_codec_add_head (io_t *io, codec_t *c)`
Insert a codec at the head the codec chain.
- `int io_codec_add_tail (io_t *io, codec_t *c)`
Append a codec to the codec chain.
- `int io_codecs_remove (io_t *io)`
Flush, remove and free all codecs in the codec chain.
- `int io_name_set (io_t *io, const char *name)`
Set the name of an io_t object.
- `int io_name_get (io_t *io, char *name, size_t sz)`
Return the name of the given io_t object.

4.6.1 Function Documentation

4.6.1.1 `enum io_type_e io_type (io_t *io)`

Return the type of the given io (see enum io_type_e).

Parameters:

io input IO object

Returns:

on of enum io_type_e defined item

Definition at line 43 of file io.c.

References io_type().

Referenced by io_type().

4.6.1.2 `ssize_t io_pipe (io_t *out, io_t *in)`

Read all data from *in* and copy it to *out*

Parameters:

out output IO object

in input IO object

Returns:

the number of bytes read from *in* and written to *out* or -1 on error

Definition at line 61 of file io.c.

References io_read(), and io_write().

4.6.1.3 `int io_dup (io_t * io, io_t ** pio)`

Create a copy of `io` and store it to `*pio`. The returned object will share the same underlying IO device, the same codecs connected to `io` and the same input and output buffers. Buffers, codecs and IO devices will not be released until all `io_t` object associated to it will be freed.

Parameters:

- io* the `io_t` object to be duffed
- pio* on success will contain the duplicated `io_t` object

Returns:

0 on success, not zero on failure

See also:

`io_free`(p. 35)

Definition at line 106 of file `io.c`.

4.6.1.4 `ssize_t io_copy (io_t * out, io_t * in, size_t size)`

Read from `in` a block of data `size` bytes long and write it to the `out` output `io_t` object

Parameters:

- out* output `io_t` object
- in* input `io_t` object
- size* number of bytes to copy

Returns:

the number of bytes copied (that can be less the `size` in case of EOF on `in`) or `-1` on error.

Definition at line 132 of file `io.c`.

References `io_read()`, and `io_write()`.

4.6.1.5 `ssize_t io_seek (io_t * io, size_t off)`

Moves the read/write file offset so that the next read or the next write will start at the given position. Note that not all `io_t` devices support seeking (e.g. sockets don't) so this function will always fail when used on those devices.

Parameters:

- io* the `io_t` object
- off* absolute offset to move to

Returns:

the given offset on success, `-1` on error

Definition at line 175 of file `io.c`.

References `io_flush()`.

4.6.1.6 `ssize_t io_tell (io_t * io)`

Return the current file position. There exists a unique read and write position offset.

Parameters:

io the `io_t` object

Returns:

the given offset on success, -1 on error

Definition at line 197 of file `io.c`.

References `io_flush()`.

4.6.1.7 `int io_close (io_t * io)`

Close the underlying source/sink of the given `io_t` object.

Parameters:

io the `io_t` object to be free'd

Returns:

0 on success, non-zero on error

See also:

`io_dup`(p. 36)

Definition at line 442 of file `io.c`.

Referenced by `io_free()`.

4.6.1.8 `int io_free (io_t * io)`

Free the given `io_t` object. If *io* has been dup'd and the reference count is not zero then this function will only decrement it and return. Otherwise *io* will be flushed, the codec applied to it (if any) freed and any other resource associated to it released.

Parameters:

io the `io_t` object to be free'd

Returns:

0 on success, non-zero on error

See also:

`io_dup`(p. 36)

Definition at line 469 of file `io.c`.

References `io_close()`, `io_codecs_remove()`, and `io_flush()`.

Referenced by `u_tmpfile_open()`.

4.6.1.9 `ssize_t io_read (io_t * io, char * buf, size_t size)`

Read `size` bytes from `io` and save them to `buf` (that must be big enough).

Parameters:

- io* the `io_t` object onto which the read operation is performed
- buf* the buffer that will contain the read bytes
- size* number of bytes to read

Returns:

the number of bytes read and saved to `buf`, 0 on end of file condition, and -1 on error.

Definition at line 572 of file `io.c`.

Referenced by `io_copy()`, `io_getc()`, `io_pipe()`, and `u_md5io()`.

4.6.1.10 `ssize_t io_vprintf (io_t * io, const char * fmt, va_list ap)`

Vprintf-like function used to easily write strings to `io` using well-known printf format strings. See `printf(3)` manual for format description.

Parameters:

- io* the `io_t` object to write to
- fmt* printf-style format string
- ap* variable list arguments

Returns:

the number of chars written on success, -1 on error

Definition at line 620 of file `io.c`.

References `io_write()`.

Referenced by `io_printf()`.

4.6.1.11 `ssize_t io_printf (io_t * io, const char * fmt, ...)`

Printf-like function used to easily write strings to `io` using well-known printf format strings. See `printf(3)` manual for format description.

Parameters:

- io* the `io_t` object to write to
- fmt* printf-style format string
- ...* format string arguments

Returns:

the number of chars written on success, -1 on error

Definition at line 691 of file `io.c`.

References `io_vprintf()`.

4.6.1.12 `ssize_t io_flush (io_t * io)`

Force a write of all buffered data to the output device.

Parameters:

io the `io_t` object to be flushed

Returns:

zero on success, -1 on error

Definition at line 721 of file `io.c`.

Referenced by `io_free()`, `io_seek()`, `io_tell()`, and `io_write()`.

4.6.1.13 `ssize_t io_write (io_t * io, const char * buf, size_t size)`

Write `size` bytes of `buf` to `io`.

Parameters:

io the `io_t` object

buf the buffer with the bytes to write

size number of bytes to write

Returns:

the number of bytes written or -1 on error.

Definition at line 756 of file `io.c`.

References `io_flush()`.

Referenced by `io_copy()`, `io_pipe()`, `io_putc()`, and `io_vprintf()`.

4.6.1.14 `ssize_t io_putc (io_t * io, char c)` [inline]

Write the character `c` to `io`

Parameters:

io the `io_t` object

c the char to write

Returns:

the number of bytes written (i.e. 1) on success or -1 on error.

Definition at line 798 of file `io.c`.

References `io_write()`.

4.6.1.15 `ssize_t io_getc (io_t * io, char * pc)` [inline]

Read a char from the `io_t` object `io` and save it at `*pc`.

Parameters:

io the `io_t` object

pc on success will hold the read character

Returns:

the number of bytes read (i.e. 1) on success or -1 on error.

Definition at line 814 of file io.c.

References io_read().

4.6.1.16 `ssize_t io_get_until (io_t * io, char stop_at, char * buf, size_t size)`

Read from *in* and save it to *buf* that must be at least *size* bytes long. Read stops when *stop_at* characted is found in the incoming stream.

Parameters:

io the io_t object

stop_at reads until this character is found

buf destination buffer

size size of buf

Returns:

the length of the line on success, 0 on EOF or -1 on error.

Definition at line 852 of file io.c.

Referenced by io_gets().

4.6.1.17 `ssize_t io_gets (io_t * io, char * buf, size_t size)`

Read a line from *in* and save it to *buf* that must be at least *size* bytes long.

Parameters:

io the io_t object

buf destination buffer

size size of buf

Returns:

the length of the line on success, 0 on EOF or -1 on error.

Definition at line 925 of file io.c.

References io_get_until().

Referenced by u_getline().

4.6.1.18 `int io_codec_add_head (io_t * io, codec_t * c)`

Parameters:

io the io_t object

c the codec to append

Returns:

0 on success, non-zero on error

Definition at line 939 of file io.c.

4.6.1.19 `int io_codec_add_tail (io_t * io, codec_t * c)`**Parameters:**

io the `io_t` object
c the codec to append

Returns:

0 on success, non-zero on error

Definition at line 959 of file `io.c`.

4.6.1.20 `int io_codecs_remove (io_t * io)`**Parameters:**

io the `io_t` object

Returns:

0 on success, non-zero on error

Definition at line 978 of file `io.c`.

References `codec_free()`.

Referenced by `io_free()`.

4.6.1.21 `int io_name_set (io_t * io, const char * name)`

Set the name of the given `io` to `name`. A name is a label that can be used to store any naming scheme (file names, URI, etc.)

Parameters:

io the `io_t` object
name the name to be given to `io`

Returns:

0 on success, non-zero on error

Definition at line 1013 of file `io.c`.

Referenced by `u_file_open()`, and `u_tmpfile_open()`.

4.6.1.22 `int io_name_get (io_t * io, char * name, size_t sz)`

Save in `name` the name of `io`.

Parameters:

io the `io_t` object
name on success will contain the name of the given `io`
sz size of `name`

Returns:

0 on success, non-zero on error

Definition at line 1044 of file `io.c`.

4.7 Filters for Compression and Encryption

Functions

- `int codec_cipher_create` (`int op`, `const EVP_CIPHER *cipher`, `unsigned char *key`, `unsigned char *iv`, `codec_t **pcc`)
Create a cipher codec_t object.
- `int codec_free` (`codec_t *codec`)
Dispose all the resources allocated to the supplied codec.
- `int codec_gzip_create` (`int op`, `codec_t **piz`)
Create a cipher codec_t object.
- `int codec_null_create` (`codec_t **pcn`)
Create a cipher codec_t object.

4.7.1 Function Documentation

4.7.1.1 `int codec_cipher_create` (`int op`, `const EVP_CIPHER * cipher`, `unsigned char * key`, `unsigned char * iv`, `codec_t ** pcc`)

Create a cipher `codec_t` object at `*pcc` suitable for encryption or decryption (depending on `op`). The `cipher`, `key` and `iv` parameters hold the algorithm, key and initialisation vector respectively, used for the data transforms.

Parameters:

op one of `CIPHER_ENCRYPT` or `CIPHER_DECRYPT`
cipher an OpenSSL `EVP_CIPHER` object
key the encryption/decryption key
iv the initialisation vector
pcc the created codec as a value-result argument

Returns:

0 on success, ~ 0 otherwise

Definition at line 172 of file `cipher.c`.

4.7.1.2 `int codec_free` (`codec_t * codec`)

Dispose all the resources allocated to the supplied `codec`

Parameters:

codec the `codec_t` object to be disposed

Returns:

always successful, i.e. 0

Definition at line 24 of file `codec.c`.

Referenced by `io_codecs_remove()`.

4.7.1.3 int codec_gzip_create (int *op*, codec_t ** *piz*)

Create a gzip codec_t object at *piz suitable for compression or decompression (depending on op).

Parameters:

op one of GZIP_COMPRESS or GZIP_UNCOMPRESS
piz the created codec as a value-result argument

Returns:

0 on success, ~0 otherwise

Definition at line 142 of file gzip.c.

4.7.1.4 int codec_null_create (codec_t ** *pcn*)

Create a null codec_t object at *pcn.

Parameters:

pcn the created codec as a value-result argument

Returns:

0 on success, ~0 otherwise

Definition at line 75 of file null.c.

4.8 Miscellaneous Utility Functions

Functions

- `int u_asctime_to_tt` (const char *str, time_t *tp)
Convert an `asctime(3)` string to `time_t`.
- `int u_rfc850_to_tt` (const char *str, time_t *tp)
Convert an `rfc850` time string to `time_t`.
- `int u_rfc822_to_tt` (const char *str, time_t *tp)
Convert an `rfc822` time string to `time_t`.
- `int u_httpdate_to_tt` (const char *str, time_t *tp)
Convert an `HTTP` time string to `time_t`.
- `int u_tt_to_rfc822` (char dst[], time_t ts)
Convert a `time_t` value to a `rfc822` time string.
- `char * u_strnstr` (const char *buf, const char *sub, size_t buflen)
Locate a substring in another string.
- `int u_foreach_dir_item` (const char *path, unsigned int mask, int(*cb)(struct dirent *, const char *, void *), void *arg)
Apply the supplied callback to each file in a given directory.
- `int u_match_ext` (const char *filename, const char *extension)
Match filename extension.
- `ssize_t u_sqlncpy` (char *d, const char *s, size_t slen, int flags)
Copy and `SQL` escape/unescape a given string.
- `ssize_t u_urlncpy` (char *d, const char *s, size_t slen, int flags)
Copy and `URL` escape/unescape a given string.
- `ssize_t u_hexncpy` (char *d, const char *s, size_t slen, int flags)
Copy and `HEX` encode/decode a given string.
- `ssize_t u_htmlncpy` (char *d, const char *s, size_t slen, int flags)
Copy and `HTML` escape/unescape a given string.
- `char * u_stristr` (const char *string, const char *sub)
Locate a given substring ignoring case.
- `char * u_strnrchr` (const char *s, char c, size_t len)
Locate a character in a string.
- `int u_tmpfile_open` (io_t **pio)
Create a temporary `io_t` object.

- `int u_file_open` (const char *file, int flags, io_t **pio)
Create an io_t object from the file system object file.
- `int u_getline` (io_t *io, u_string_t *ln)
Read a line from the io_t object io.
- `int u_fgetline` (FILE *in, u_string_t *ln)
get a line from a FILE object
- `int u_file_exists` (const char *fqfn)
Tell if the given file exists.
- `void u_tohex` (char *hex, const char *src, size_t sz)
Convert a given string in hexadecimal representation.
- `int u_md5` (const char *buf, size_t sz, char out[MD5_DIGEST_BUFSZ])
Calculate the MD5 digest over a given buffer.
- `int u_md5io` (io_t *io, char out[MD5_DIGEST_BUFSZ])
Calculate the MD5 hash over an io_t stream.
- `const mime_map_t * u_get_mime_map` (const char *file_name)
Get the MIME type of a file.
- `const char * u_guess_mime_type` (const char *file_name)
Guess the MIME type of a file.
- `const char * kclone_version` (void)
Return KClone version string (x.y.z).

4.8.1 Detailed Description

4.8.2 Function Documentation

4.8.2.1 `int u_asctime_to_tt` (const char * str, time_t * tp)

Convert the `asctime(3)` string `str` to its `time_t` representation `tp`.

Parameters:

str the string to be converted

tp the `time_t` conversion of `str` as a value-result argument

Returns:

- 0 successful

- ~0 failure

Definition at line 66 of file date.c.

Referenced by `u_httpdate_to_tt()`.

4.8.2.2 `int u_rfc850_to_tt (const char * str, time_t * tp)`

Convert the rfc850 string `str` to its `time_t` representation `tp`.

Parameters:

str the string to be converted

tp the `time_t` conversion of `str` as a value-result argument

Returns:

- 0 successful
- ~0 failure

Definition at line 112 of file date.c.

Referenced by `u_httpdate_to_tt()`.

4.8.2.3 `int u_rfc822_to_tt (const char * str, time_t * tp)`

Convert the rfc822 string `str` to its `time_t` representation `tp`.

Parameters:

str the string to be converted

tp the `time_t` conversion of `str` as a value-result argument

Returns:

- 0 successful
- ~0 failure

Definition at line 164 of file date.c.

Referenced by `u_httpdate_to_tt()`.

4.8.2.4 `int u_httpdate_to_tt (const char * str, time_t * tp)`

Convert the HTTP time string `str` to its `time_t` representation `tp`.

Parameters:

str the string to be converted

tp the `time_t` conversion of `str` as a value-result argument

Returns:

- 0 successful
- ~0 failure

Definition at line 214 of file date.c.

References `u_asctime_to_tt()`, `u_rfc822_to_tt()`, and `u_rfc850_to_tt()`.

4.8.2.5 `int u_tt_to_rfc822 (char dst[], time_t ts)`

Convert the `time_t` value `ts` to a rfc822 time string

Parameters:

- dst* placeholder for the rfc822 time string. The buffer, of at least `RFC822_DATE_BUFSZ` bytes, must be preallocated by the caller.
- ts* the `time_t` value to be converted

Returns:

- 0 successful
- ~0 failure

Definition at line 242 of file `date.c`.

Referenced by `response_set_cookie()`, `response_set_date()`, and `response_set_last_modified()`.

4.8.2.6 `char* u_strnstr (const char * buf, const char * sub, size_t buflen)`

The function locates the first occurrence of `sub` in the string `buf` of size `buflen`

Parameters:

- buf*
- sub*
- buflen*

Returns:

- 0 successful
- ~0 error

Definition at line 93 of file `utils.c`.

4.8.2.7 `int u_foreach_dir_item (const char * path, unsigned int mask, int(* cb)(struct dirent *, const char *, void *), void * arg)`

Apply the supplied callback `cb` with additional arguments `arg` to each file in directory `path` which match the given `mask`.

Parameters:

- path* directory path
- mask* matching file mask
- cb* function to call
- arg* optional additional arguments

Returns:

- 0 successful
- ~0 error

Definition at line 133 of file `utils.c`.

References `u_foreach_dir_item()`.

Referenced by `u_foreach_dir_item()`.

4.8.2.8 `int u_match_ext (const char * filename, const char * extension)`

Return 1 if the filename extension is equal to `extension` (case-insensitive comparison).

Parameters:

filename file name
extension file extension to match

Returns:

- 1 if `filename` extension is `extension`
- 0 if `filename` extension is not equal to `extension`

Definition at line 186 of file `utils.c`.

4.8.2.9 `ssize_t u_sqlncpy (char * d, const char * s, size_t slen, int flags)`

Copy and SQL escape/unescape, depending on `flags` value, the string `s` into `d`. The destination string, which must be at least `slen + 1` bytes long, is NULL terminated.

Parameters:

d the encoded/decoded string
s string to process
slen length of `s`
flags one of `SQLCPY_ENCODE` or `SQLCPY_DECODE`

Returns:

The number of characters written to `d` not including the trailing `'\0'` or `-1` on error.

Definition at line 292 of file `utils.c`.

4.8.2.10 `ssize_t u_urlncpy (char * d, const char * s, size_t slen, int flags)`

Copy an URL escaped/unescaped version of string `s`, depending on `flags` value, into `d`. The destination string is NULL terminated. The destination string `d` must be at least `slen + 1` bytes long.

Parameters:

d the encoded/decoded string
s string to process
slen length of `s`
flags one of `URLCPY_ENCODE` or `URLCPY_DECODE`

Returns:

The number of characters written to `d` not including the trailing `'\0'` or `-1` on error.

Definition at line 391 of file `utils.c`.

Referenced by `response_set_cookie()`.

4.8.2.11 `ssize_t u_hexncpy (char * d, const char * s, size_t slen, int flags)`

Copy an HEX encoded/decoded version of string `s`, depending on `flags` value, into `d`. The destination string `d`, which must be at least `slen + 1` bytes long, is NULL terminated.

Parameters:

- `d` the encoded/decoded string
- `s` string to process
- `slen` length of `s`
- `flags` one of `HEXCPY_ENCODE` or `HEXCPY_DECODE`

Returns:

The number of characters written to `d` not including the trailing `'\0'` or `-1` on error.

Definition at line 495 of file `utils.c`.

4.8.2.12 `ssize_t u_htmlncpy (char * d, const char * s, size_t slen, int flags)`

Copy an HTML escaped/unescaped version of string `s`, depending on `flags` value, into `d`. The destination string is NULL terminated. The destination string `d` must be at least `slen + 1` bytes long.

Parameters:

- `d` the encoded/decoded string
- `s` string to process
- `slen` length of `s`
- `flags` one of `HTMLCPY_ENCODE` or `HTMLCPY_DECODE`

Returns:

The number of characters written to `d` not including the trailing `'\0'` or `-1` on error.

Definition at line 593 of file `utils.c`.

4.8.2.13 `char* u_stristr (const char * string, const char * sub)`

Locate the first occurrence of the null-terminated string `sub` in the null-terminated string `string`, ignoring the case of both string.

Parameters:

- `string` string to be searched
- `sub` substring to search

Returns:

the pointer to the found substring or NULL if `sub` occurs nowhere in `string`

Definition at line 626 of file `utils.c`.

4.8.2.14 `char* u_strnrchr (const char * s, char c, size_t len)`

Locate the last occurrence of `c` in the substring of length `len` starting at `s`.

Parameters:

- `s` pointer to the starting of the string
- `c` the character to search
- `len` length of the string to be searched

Returns:

the pointer to the character, or NULL if `c` doesn't occur in `s`

Definition at line 658 of file `utils.c`.

4.8.2.15 `int u_tmpfile_open (io_t ** pio)`

Create a temporary `io_t` object at `*pio`.

Parameters:

- `pio` pointer to the temporary `io_t` object

Returns:

- 0 successful
- ~0 error

Definition at line 683 of file `utils.c`.

References `io_free()`, `io_name_set()`, and `u_file_open()`.

4.8.2.16 `int u_file_open (const char * file, int flags, io_t ** pio)`

Create an `io_t` object at `*pio` from the file system object `file`. The file is opened with the permission bits given in `mode`.

Parameters:

- `file` pathname of the file to open
- `flags` permission bits passed to the open syscall
- `pio` the `io_t` object associated to `file`

Returns:

- 0 successful
- ~0 error

Definition at line 722 of file `utils.c`.

References `io_name_set()`.

Referenced by `u_tmpfile_open()`.

4.8.2.17 `int u_getline (io_t * io, u_string_t * ln)`

Read a line and place it into `ln` from the `io_t` object `io`

Parameters:

- `io` an initialised `io_t` object
- `ln` the line read

Returns:

- 0 successful
- ~0 error

Definition at line 764 of file `utils.c`.

References `io_gets()`.

4.8.2.18 `int u_fgetline (FILE * in, u_string_t * ln)`

Try to get a line from the `FILE` object `in` and store it at `ln`.

Parameters:

- `in` the `FILE` object from which read is performed
- `ln` the `u_string_t` object where the line read is stored

Returns:

- 0 successful
- ~0 error

Definition at line 803 of file `utils.c`.

4.8.2.19 `int u_file_exists (const char * fqn)`

Tell if the given file `fqn` exists

Parameters:

- `fqn` the path of the (regular) file to check

Returns:

- 1 if the file exists and is a regular file, 0 otherwise

Definition at line 888 of file `utils.c`.

4.8.2.20 `void u_tohex (char * hex, const char * src, size_t sz)`

Convert the string `src` of length `sz` into its hexadecimal representation `hex`. The string `hex` must be at least `2 * sz` long.

Parameters:

- `hex` the string holding the hexadecimal conversion of `src`
- `src` the string that has to be converted

sz the length of *src*

Returns:

nothing

Definition at line 911 of file `utils.c`.

Referenced by `u_md5()`, and `u_md5io()`.

4.8.2.21 `int u_md5 (const char * buf, size_t sz, char out[MD5_DIGEST_BUFSZ])`

Calculate the MD5 digest over the supplied buffer *buf* of size *sz* and place it at *out*.

Parameters:

buf the buffer to be hashed

sz length in bytes of *buf*

out hexadecimal string containing the MD5 hash calculated over *buf*. It must be at least `MD5_DIGEST_BUFSZ` bytes long.

Returns:

- 0 always successful

Definition at line 942 of file `utils.c`.

References `u_tohex()`.

4.8.2.22 `int u_md5io (io_t * io, char out[MD5_DIGEST_BUFSZ])`

Calculate the MD5 hash over an `io_t` stream *io* and place the result as an hexadecimal string into *out*.

Parameters:

io the `io_t` stream to be hashed

out hexadecimal string containing the MD5 hash calculated over *buf*. It must be at least `MD5_DIGEST_BUFSZ` bytes long.

Returns:

- 0 successful
- ~ 0 error

Definition at line 975 of file `utils.c`.

References `io_read()`, and `u_tohex()`.

4.8.2.23 `const mime_map_t* u_get_mime_map (const char * file_name)`

Get the MIME type of the given file *file_name* by its extension

Parameters:

file_name the path of the file

Returns:

the found MIME map, or the first map if no match could be found

Definition at line 1037 of file `utils.c`.

4.8.2.24 `const char* u_guess_mime_type (const char * file_name)`

Guess the MIME type of the given file `file_name` by its extension

Parameters:

file_name the path of the file

Returns:

the string corresponding to the guessed MIME type, or "application/octet-stream" in case no map could be found

Definition at line 1070 of file `utils.c`.

4.8.2.25 `const char* kclone_version (void)`

Return KClone version string in the format `x.y.z`.

Returns:

the version string

Definition at line 30 of file `version.c`.

Chapter 5

KLone File Documentation

5.1 http.h File Reference

Enumerations

- enum {
HTTP_STATUS_EMPTY = 0, HTTP_STATUS_OK = 200, HTTP_STATUS_CREATED = 201, HTTP_STATUS_ACCEPTED = 202,
HTTP_STATUS_NO_CONTENT = 204, HTTP_STATUS_MOVED_PERMANENTLY = 301, HTTP_STATUS_MOVED_TEMPORARILY = 302,
HTTP_STATUS_NOT_MODIFIED = 304,
HTTP_STATUS_BAD_REQUEST = 400, HTTP_STATUS_UNAUTHORIZED = 401, HTTP_STATUS_FORBIDDEN = 403, HTTP_STATUS_NOT_FOUND = 404,
HTTP_STATUS_REQUEST_TIMEOUT = 408, HTTP_STATUS_LENGTH_REQUIRED = 411, HTTP_STATUS_REQUEST_TOO_LARGE = 413, HTTP_STATUS_INTERNAL_SERVER_ERROR = 500,
HTTP_STATUS_NOT_IMPLEMENTED = 501, HTTP_STATUS_BAD_GATEWAY = 502, HTTP_STATUS_SERVICE_UNAVAILABLE = 503 }
• enum http_method_e {
HM_UNKNOWN, HM_GET, HM_HEAD, HM_POST,
HM_PUT, HM_DELETE }

5.1.1 Detailed Description

Definition in file `http.h`.

5.1.2 Enumeration Type Documentation

5.1.2.1 anonymous enum

HTTP response codes

Enumeration values:

HTTP_STATUS_EMPTY undefined status
HTTP_STATUS_OK request succeeded
HTTP_STATUS_CREATED fulfilled request resulting in creation of new resource
HTTP_STATUS_ACCEPTED request accepted but processing not completed
HTTP_STATUS_NO_CONTENT no body returned
HTTP_STATUS_MOVED_PERMANENTLY resource relocated permanently
HTTP_STATUS_MOVED_TEMPORARILY resource relocated temporarily
HTTP_STATUS_NOT_MODIFIED GET request for unmodified document
HTTP_STATUS_BAD_REQUEST syntax error
HTTP_STATUS_UNAUTHORIZED user authentication required
HTTP_STATUS_FORBIDDEN access to resource forbidden
HTTP_STATUS_NOT_FOUND request timeout
HTTP_STATUS_REQUEST_TIMEOUT nothing found at matching request URI
HTTP_STATUS_LENGTH_REQUIRED missing Content-Length header field
HTTP_STATUS_REQUEST_TOO_LARGE request PDU too big
HTTP_STATUS_INTERNAL_SERVER_ERROR unexpected condition caused an error
HTTP_STATUS_NOT_IMPLEMENTED request method not supported
HTTP_STATUS_BAD_GATEWAY invalid response while acting as gateway or proxy
HTTP_STATUS_SERVICE_UNAVAILABLE server unavailable due to temporary overloading or maintenance

Definition at line 23 of file http.h.

5.1.2.2 enum http_method_e

HTTP Methods

Enumeration values:

HM_UNKNOWN unknown value
HM_GET retrieve data at URI
HM_HEAD ~HM_GET with headers only
HM_POST create new object subordinate to specified object
HM_PUT data in body is to be stored under URL
HM_DELETE deletion request at given URL

Definition at line 65 of file http.h.

5.2 response.h File Reference

Functions

- int **response_redirect** (response_t *rs, const char *url)
Redirect to a given url.
- int **response_set_status** (response_t *rs, int code)
Set the status of a response.
- int **response_get_status** (response_t *rs)
Get the status of a response.
- void **response_set_method** (response_t *rs, int method)
Set the response method.
- int **response_get_method** (response_t *rs)
Get the response method.
- int **response_enable_caching** (response_t *rs)
Remove all headers that inhibit page caching.
- int **response_disable_caching** (response_t *rs)
Add all header field that enable page caching (i.e. disable caching).
- int **response_print_header** (response_t *rs)
Print a response header.
- io_t * **response_io** (response_t *rs)
Get the I/O object of a response.
- header_t * **response_get_header** (response_t *rs)
Get the header of a response.
- int **response_set_field** (response_t *rs, const char *name, const char *value)
Set an header field of a response object.
- int **response_set_content_type** (response_t *rs, const char *mime_type)
Set the content type of a response to a mime type.
- int **response_set_content_length** (response_t *rs, size_t sz)
Set the content length field of a response header.
- int **response_set_content_encoding** (response_t *rs, const char *encoding)
Set response content encoding field.
- int **response_set_last_modified** (response_t *rs, time_t mtime)
Set the last modified field in a response header.
- int **response_set_date** (response_t *rs, time_t now)

Set the date field in a response header.

- int **response_set_cookie** (response_t *rs, const char *name, const char *value, time_t expire, const char *path, const char *domain, int secure)

Set the value of a cookie.

5.2.1 Detailed Description

Definition in file **response.h**.

Index

- basic
 - io_close, 33
 - io_codec_add_head, 36
 - io_codec_add_tail, 36
 - io_codecs_remove, 37
 - io_copy, 32
 - io_dup, 31
 - io_flush, 34
 - io_free, 33
 - io_get_until, 36
 - io_getc, 35
 - io_gets, 36
 - io_name_get, 37
 - io_name_set, 37
 - io_pipe, 31
 - io_printf, 34
 - io_putc, 35
 - io_read, 33
 - io_seek, 32
 - io_tell, 32
 - io_type, 31
 - io_vprintf, 34
 - io_write, 35
- Basic Functions, 30
- codec_cipher_create
 - filters, 38
- codec_free
 - filters, 38
- codec_gzip_create
 - filters, 38
- codec_null_create
 - filters, 39
- Dynamic page interfaces, 9
- filters
 - codec_cipher_create, 38
 - codec_free, 38
 - codec_gzip_create, 38
 - codec_null_create, 39
- Filters for Compression and Encryption, 38
- HM_DELETE
 - http.h, 52
- HM_GET
 - http.h, 52
- HM_HEAD
 - http.h, 52
- HM_POST
 - http.h, 52
- HM_PUT
 - http.h, 52
- HM_UNKNOWN
 - http.h, 52
- http.h, 51
 - HM_DELETE, 52
 - HM_GET, 52
 - HM_HEAD, 52
 - HM_POST, 52
 - HM_PUT, 52
 - HM_UNKNOWN, 52
 - http_method_e, 52
 - HTTP_STATUS_ACCEPTED, 52
 - HTTP_STATUS_BAD_GATEWAY, 52
 - HTTP_STATUS_BAD_REQUEST, 52
 - HTTP_STATUS_CREATED, 52
 - HTTP_STATUS_EMPTY, 52
 - HTTP_STATUS_FORBIDDEN, 52
 - HTTP_STATUS_INTERNAL_SERVER_ERROR, 52
 - HTTP_STATUS_LENGTH_REQUIRED, 52
 - HTTP_STATUS_MOVED_PERMANENTLY, 52
 - HTTP_STATUS_MOVED_TEMPORARILY, 52
 - HTTP_STATUS_NO_CONTENT, 52
 - HTTP_STATUS_NOT_FOUND, 52
 - HTTP_STATUS_NOT_IMPLEMENTED, 52
 - HTTP_STATUS_NOT_MODIFIED, 52
 - HTTP_STATUS_OK, 52
 - HTTP_STATUS_REQUEST_TIMEOUT, 52
 - HTTP_STATUS_REQUEST_TOO_LARGE, 52
 - HTTP_STATUS_SERVICE_UNAVAILABLE, 52
 - HTTP_STATUS_UNAUTHORIZED, 52
- http_method_e

- http.h, 52
- HTTP_STATUS_ACCEPTED
 - http.h, 52
- HTTP_STATUS_BAD_GATEWAY
 - http.h, 52
- HTTP_STATUS_BAD_REQUEST
 - http.h, 52
- HTTP_STATUS_CREATED
 - http.h, 52
- HTTP_STATUS_EMPTY
 - http.h, 52
- HTTP_STATUS_FORBIDDEN
 - http.h, 52
- HTTP_STATUS_INTERNAL_SERVER_ERROR
 - http.h, 52
- HTTP_STATUS_LENGTH_REQUIRED
 - http.h, 52
- HTTP_STATUS_MOVED_PERMANENTLY
 - http.h, 52
- HTTP_STATUS_MOVED_TEMPORARILY
 - http.h, 52
- HTTP_STATUS_NO_CONTENT
 - http.h, 52
- HTTP_STATUS_NOT_FOUND
 - http.h, 52
- HTTP_STATUS_NOT_IMPLEMENTED
 - http.h, 52
- HTTP_STATUS_NOT_MODIFIED
 - http.h, 52
- HTTP_STATUS_OK
 - http.h, 52
- HTTP_STATUS_REQUEST_TIMEOUT
 - http.h, 52
- HTTP_STATUS_REQUEST_TOO_LARGE
 - http.h, 52
- HTTP_STATUS_SERVICE_UNAVAILABLE
 - http.h, 52
- HTTP_STATUS_UNAUTHORIZED
 - http.h, 52
- Input/Output, 29
- io_close
 - basic, 33
- io_codec_add_head
 - basic, 36
- io_codec_add_tail
 - basic, 36
- io_codecs_remove
 - basic, 37
- io_copy
 - basic, 32
- io_dup
 - basic, 31
- io_flush
 - basic, 34
- io_free
 - basic, 33
- io_get_until
 - basic, 36
- io_getc
 - basic, 35
- io_gets
 - basic, 36
- io_name_get
 - basic, 37
- io_name_set
 - basic, 37
- io_pipe
 - basic, 31
- io_printf
 - basic, 34
- io_putc
 - basic, 35
- io_read
 - basic, 33
- io_seek
 - basic, 32
- io_tell
 - basic, 32
- io_type
 - basic, 31
- io_vprintf
 - basic, 34
- io_write
 - basic, 35
- klone_version
 - ut, 49
- Miscellaneous Utility Functions, 40
- request
 - request_get_addr, 18
 - request_get_arg, 13
 - request_get_args, 12
 - request_get_client_request, 15
 - request_get_content_length, 15
 - request_get_cookie, 12
 - request_get_cookies, 12
 - request_get_field, 18
 - request_get_field_value, 19
 - request_get_filename, 14
 - request_get_getarg, 13
 - request_get_getargs, 12
 - request_get_header, 18

- request_get_http, 15
- request_get_if_modified_since, 15
- request_get_method, 17
- request_get_path_info, 14
- request_get_peer_addr, 18
- request_get_postarg, 13
- request_get_postargs, 13
- request_get_protocol, 17
- request_get_query_string, 14
- request_get_resolved_filename, 17
- request_get_resolved_path_info, 17
- request_get_uploaded_file, 16
- request_get_uploads, 16
- request_get_uri, 14
- request_io, 11
- Request Handling, 10
- request_get_addr
 - request, 18
- request_get_arg
 - request, 13
- request_get_args
 - request, 12
- request_get_client_request
 - request, 15
- request_get_content_length
 - request, 15
- request_get_cookie
 - request, 12
- request_get_cookies
 - request, 12
- request_get_field
 - request, 18
- request_get_field_value
 - request, 19
- request_get_filename
 - request, 14
- request_get_getarg
 - request, 13
- request_get_getargs
 - request, 12
- request_get_header
 - request, 18
- request_get_http
 - request, 15
- request_get_if_modified_since
 - request, 15
- request_get_method
 - request, 17
- request_get_path_info
 - request, 14
- request_get_peer_addr
 - request, 18
- request_get_postarg
 - request, 13
- request_get_postargs
 - request, 13
- request_get_protocol
 - request, 17
- request_get_query_string
 - request, 14
- request_get_resolved_filename
 - request, 17
- request_get_resolved_path_info
 - request, 17
- request_get_uploaded_file
 - request, 16
- request_get_uploads
 - request, 16
- request_get_uri
 - request, 14
- request_io
 - request, 11
- response
 - response_del_field, 23
 - response_disable_caching, 21
 - response_enable_caching, 21
 - response_get_header, 25
 - response_get_method, 22
 - response_get_status, 25
 - response_io, 25
 - response_print_header, 23
 - response_redirect, 25
 - response_set_content_encoding, 21
 - response_set_content_length, 24
 - response_set_content_type, 24
 - response_set_cookie, 22
 - response_set_date, 24
 - response_set_field, 23
 - response_set_last_modified, 24
 - response_set_method, 22
 - response_set_status, 26
- Response Handling, 20
- response.h, 53
- response_del_field
 - response, 23
- response_disable_caching
 - response, 21
- response_enable_caching
 - response, 21
- response_get_header
 - response, 25
- response_get_method
 - response, 22
- response_get_status
 - response, 25
- response_io
 - response, 25
- response_print_header

- response, 23
- response_redirect
 - response, 25
- response_set_content_encoding
 - response, 21
- response_set_content_length
 - response, 24
- response_set_content_type
 - response, 24
- response_set_cookie
 - response, 22
- response_set_date
 - response, 24
- response_set_field
 - response, 23
- response_set_last_modified
 - response, 24
- response_set_method
 - response, 22
- response_set_status
 - response, 26
- session
 - session_age, 28
 - session_clean, 28
 - session_del, 28
 - session_get, 27
 - session_get_vars, 27
 - session_set, 27
- session_age
 - session, 28
- session_clean
 - session, 28
- session_del
 - session, 28
- session_get
 - session, 27
- session_get_vars
 - session, 27
- session_set
 - session, 27
- Sessions, 27
- u_asctime_to_tt
 - ut, 41
- u_fgetline
 - ut, 47
- u_file_exists
 - ut, 47
- u_file_open
 - ut, 46
- u_foreach_dir_item
 - ut, 43
- u_get_mime_map
 - ut, 48
- u_getline
 - ut, 46
- u_guess_mime_type
 - ut, 48
- u_hexncpy
 - ut, 44
- u_htmlncpy
 - ut, 45
- u_httpdate_to_tt
 - ut, 42
- u_match_ext
 - ut, 43
- u_md5
 - ut, 48
- u_md5io
 - ut, 48
- u_rfc822_to_tt
 - ut, 42
- u_rfc850_to_tt
 - ut, 42
- u_sqlncpy
 - ut, 44
- u_stristr
 - ut, 45
- u_strnrchr
 - ut, 45
- u_strnstr
 - ut, 43
- u_tmpfile_open
 - ut, 46
- u_tohex
 - ut, 47
- u_tt_to_rfc822
 - ut, 42
- u_urlncpy
 - ut, 44
- ut
 - klone_version, 49
 - u_asctime_to_tt, 41
 - u_fgetline, 47
 - u_file_exists, 47
 - u_file_open, 46
 - u_foreach_dir_item, 43
 - u_get_mime_map, 48
 - u_getline, 46
 - u_guess_mime_type, 48
 - u_hexncpy, 44
 - u_htmlncpy, 45
 - u_httpdate_to_tt, 42
 - u_match_ext, 43
 - u_md5, 48
 - u_md5io, 48
 - u_rfc822_to_tt, 42

u_rfc850_to_tt, 42
u_sqlncpy, 44
u_stristr, 45
u_strnrchr, 45
u_strnstr, 43
u_tmpfile_open, 46
u_tohex, 47
u_tt_to_rfc822, 42
u_urlncpy, 44