

Sagui library

v1.2.0

Generated by Doxygen 1.8.15

Mon Oct 1 2018 09:29:04

Contents

1	Main Page	2
2	Module Index	2
2.1	Modules	2
3	Data Structure Index	2
3.1	Data Structures	2
4	File Index	3
4.1	File List	3
5	Module Documentation	3
5.1	API reference	3
5.1.1	Detailed Description	3
5.2	Utilities	4
5.2.1	Detailed Description	4
5.2.2	Typedef Documentation	4
5.2.3	Function Documentation	6
5.3	String	10
5.3.1	Detailed Description	10
5.3.2	Function Documentation	10
5.4	String map	15
5.4.1	Detailed Description	15
5.4.2	Typedef Documentation	15
5.4.3	Function Documentation	16
5.5	HTTP server	24
5.5.1	Detailed Description	25
5.5.2	Macro Definition Documentation	25
5.5.3	Typedef Documentation	26
5.5.4	Function Documentation	28
5.6	Path routing	59
5.6.1	Detailed Description	60
5.6.2	Typedef Documentation	60
5.6.3	Function Documentation	62

6	Data Structure Documentation	78
6.1	sg_entrpoint Struct Reference	78
6.1.1	Detailed Description	78
6.2	sg_entrpoints Struct Reference	78
6.2.1	Detailed Description	78
6.3	sg_httpauth Struct Reference	78
6.3.1	Detailed Description	78
6.4	sg_httpreq Struct Reference	79
6.4.1	Detailed Description	79
6.5	sg_httpsres Struct Reference	79
6.5.1	Detailed Description	79
6.6	sg_httpsrv Struct Reference	79
6.6.1	Detailed Description	79
6.7	sg_httpupld Struct Reference	80
6.7.1	Detailed Description	80
6.8	sg_route Struct Reference	80
6.8.1	Detailed Description	80
6.9	sg_router Struct Reference	80
6.9.1	Detailed Description	80
6.10	sg_str Struct Reference	81
6.10.1	Detailed Description	81
6.11	sg_strmap Struct Reference	81
6.11.1	Detailed Description	81

7	File Documentation	82
7.1	example_entrypoint.h File Reference	82
7.2	example_httpauth.h File Reference	82
7.3	example_httpcookie.h File Reference	82
7.4	example_httpsrv.h File Reference	82
7.5	example_httpsrv_benchmark.h File Reference	82
7.6	example_httpsrv_tls.h File Reference	82
7.7	example_httpsrv_tls_cert_auth.h File Reference	82
7.8	example_httputlds.h File Reference	82
7.9	example_router_segments.h File Reference	82
7.10	example_router_simple.h File Reference	82
7.11	example_router_srv.h File Reference	82
7.12	example_router_vars.h File Reference	82
7.13	example_str.h File Reference	82
7.14	example_strmap.h File Reference	82
7.15	sagui.h File Reference	82
7.15.1	Macro Definition Documentation	85
8	Example Documentation	86
8.1	example_entrypoint.c	86
8.2	example_httpauth.c	87
8.3	example_httpcookie.c	88
8.4	example_httpsrv.c	89
8.5	example_httpsrv_benchmark.c	90
8.6	example_httpsrv_tls.c	91
8.7	example_httpsrv_tls_cert_auth.c	92
8.8	example_httputlds.c	95
8.9	example_router_segments.c	96
8.10	example_router_simple.c	97
8.11	example_router_srv.c	98
8.12	example_router_vars.c	99
8.13	example_str.c	100
8.14	example_strmap.c	101

Index	103
1 Main Page	
• API reference	
2 Module Index	
2.1 Modules	
Here is a list of all modules:	
API reference	3
Utilities	4
String	10
String map	15
HTTP server	24
Path routing	59
3 Data Structure Index	
3.1 Data Structures	
Here are the data structures with brief descriptions:	
sg_entrypoint	78
sg_entrypoints	78
sg_httpauth	78
sg_httpreq	79
sg_httpres	79
sg_httpsrv	79
sg_httpupld	80
sg_route	80
sg_router	80
sg_str	81
sg_strmap	81

4 File Index

4.1 File List

Here is a list of all files with brief descriptions:

example_entrypoint.h	82
example_httpauth.h	82
example_httpcookie.h	82
example_httpsrv.h	82
example_httpsrv_benchmark.h	82
example_httpsrv_tls.h	82
example_httpsrv_tls_cert_auth.h	82
example_httpuplds.h	82
example_router_segments.h	82
example_router_simple.h	82
example_router_srv.h	82
example_router_vars.h	82
example_str.h	82
example_strmap.h	82
sagui.h	82

5 Module Documentation

5.1 API reference

Modules

- [Utilities](#)
- [String](#)
- [String map](#)
- [HTTP server](#)
- [Path routing](#)

5.1.1 Detailed Description

The API reference grouped by feature.

5.2 Utilities

Typedefs

- typedef void(* [sg_err_cb](#)) (void *cls, const char *err)
- typedef size_t(* [sg_write_cb](#)) (void *handle, uint64_t offset, const char *buf, size_t size)
- typedef ssize_t(* [sg_read_cb](#)) (void *handle, uint64_t offset, char *buf, size_t size)
- typedef void(* [sg_free_cb](#)) (void *handle)
- typedef int(* [sg_save_cb](#)) (void *handle, bool overwritten)
- typedef int(* [sg_save_as_cb](#)) (void *handle, const char *path, bool overwritten)

Functions

- unsigned int [sg_version](#) (void)
- const char * [sg_version_str](#) (void)
- void * [sg_alloc](#) (size_t size) __attribute__((malloc))
- void * [sg_realloc](#) (void *ptr, size_t size) __attribute__((malloc))
- void [sg_free](#) (void *ptr)
- char * [sg_strerror](#) (int errnum, char *errmsg, size_t errlen)
- bool [sg_is_post](#) (const char *method)
- char * [sg_extract_entrypoint](#) (const char *path)
- char * [sg_tmpdir](#) (void)

5.2.1 Detailed Description

All utility functions of the library.

5.2.2 Typedef Documentation

5.2.2.1 [sg_err_cb](#)

```
typedef void(* sg_err_cb) (void *cls, const char *err)
```

Callback signature used by functions that handle errors.

Parameters

out	<i>cls</i>	User-defined closure.
out	<i>err</i>	Error message.

5.2.2.2 [sg_write_cb](#)

```
typedef size_t(* sg_write_cb) (void *handle, uint64_t offset, const char *buf, size_t size)
```

Callback signature used by functions that write streams.

Parameters

out	<i>handle</i>	Stream handle.
out	<i>offset</i>	Current stream offset.
out	<i>buf</i>	Current buffer to be written.
out	<i>size</i>	Size of the current buffer to be written.

Returns

Total written buffer.

5.2.2.3 sg_read_cb

```
typedef ssize_t(* sg_read_cb) (void *handle, uint64_t offset, char *buf, size_t size)
```

Callback signature used by functions that read streams.

Parameters

out	<i>handle</i>	Stream handle.
out	<i>offset</i>	Current stream offset.
out	<i>buf</i>	Current read buffer.
out	<i>size</i>	Size of the current read buffer.

Returns

Total read buffer.

5.2.2.4 sg_free_cb

```
typedef void(* sg_free_cb) (void *handle)
```

Callback signature used by functions that free streams.

Parameters

out	<i>handle</i>	Stream handle.
-----	---------------	----------------

5.2.2.5 sg_save_cb

```
typedef int(* sg_save_cb) (void *handle, bool overwritten)
```

Callback signature used by functions that save streams.

Parameters

out	<i>handle</i>	Stream handle.
out	<i>overwritten</i>	Overwrite an already existed stream.

Return values

0	- Success.
<i>E<ERROR></i>	- User-defined error to abort the saving.

5.2.2.6 sg_save_as_cb

```
typedef int(* sg_save_as_cb) (void *handle, const char *path, bool overwritten)
```

Callback signature used by functions that save streams. It allows to specify the destination file path.

Parameters

out	<i>handle</i>	Stream handle.
out	<i>path</i>	Absolute path to store the stream.
out	<i>overwritten</i>	Overwrite an already existed stream.

Return values

0	- Success.
<i>E<ERROR></i>	- User-defined error to abort the saving.

5.2.3 Function Documentation**5.2.3.1 sg_version()**

```
unsigned int sg_version (
    void )
```

Returns the library version number.

Returns

Library version packed into a single integer.

5.2.3.2 sg_version_str()

```
const char* sg_version_str (
    void )
```

Returns the library version number as string.

Returns

Library version packed into a null-terminated string.

5.2.3.3 sg_alloc()

```
void* sg_alloc (
    size_t size )
```

Allocates a new zero-initialize memory space.

Parameters

in	size	Memory size to be allocated.
----	------	------------------------------

Returns

Pointer of the zero-initialized allocated memory.

Return values

NULL	If size is 0 or no memory space.
------	----------------------------------

Examples:

[example_httpsrv_tls_cert_auth.c](#).

5.2.3.4 sg_realloc()

```
void* sg_realloc (
    void * ptr,
    size_t size )
```

Reallocates an existing memory block.

Parameters

in, out	ptr	Pointer of the memory to be reallocated.
in	size	Memory size to be reallocated.

Returns

Pointer of the reallocated memory.

Note

Equivalent to `realloc(3)`.

5.2.3.5 sg_free()

```
void sg_free (
    void * ptr )
```

Frees a memory space previously allocated by `sg_alloc()` or `sg_realloc()`.

Parameters

in	<i>ptr</i>	Pointer of the memory to be freed.
----	------------	------------------------------------

Examples:

[example_httpsrv_tls_cert_auth.c](#).

5.2.3.6 sg_strerror()

```
char* sg_strerror (
    int errnum,
    char * errmsg,
    size_t errlen )
```

Returns string describing an error number.

Parameters

in	<i>errnum</i>	Error number.
in, out	<i>errmsg</i>	Pointer of a string to store the error message.
in	<i>errlen</i>	Length of the error message.

Returns

Pointer to **str**.

Examples:

[example_httpsrv_tls_cert_auth.c](#), and [example_httpuplds.c](#).

5.2.3.7 sg_is_post()

```
bool sg_is_post (
    const char * method )
```

Checks if a string is a HTTP post method.

Parameters

in	<i>method</i>	Null-terminated string.
----	---------------	-------------------------

Returns

true if **method** is POST, PUT, DELETE or OPTIONS.

5.2.3.8 sg_extract_entrypoint()

```
char* sg_extract_entrypoint (
    const char * path )
```

5.2.3.9 sg_tmpdir()

```
char* sg_tmpdir (
    void )
```

Returns the system temporary directory.

Returns

Temporary directory as null-terminated string.

Return values

<i>NULL</i>	If no memory space is available.
-------------	----------------------------------

Examples:

[example_httpuplds.c](#).

5.3 String

Data Structures

- struct [sg_str](#)

Functions

- struct [sg_str](#) * [sg_str_new](#) (void) `__attribute__((malloc))`
- void [sg_str_free](#) (struct [sg_str](#) *str)
- int [sg_str_write](#) (struct [sg_str](#) *str, const char *val, size_t len)
- int [sg_str_printf_va](#) (struct [sg_str](#) *str, const char *fmt, va_list ap)
- int [sg_str_printf](#) (struct [sg_str](#) *str, const char *fmt,...) `__attribute__((format(printf`
- int const char * [sg_str_content](#) (struct [sg_str](#) *str)
- size_t [sg_str_length](#) (struct [sg_str](#) *str)
- int [sg_str_clear](#) (struct [sg_str](#) *str)

5.3.1 Detailed Description

String handle and its related functions.

5.3.2 Function Documentation

5.3.2.1 [sg_str_new\(\)](#)

```
struct sg\_str* sg\_str\_new (  
    void )
```

Creates a new zero-initialized string handle.

Returns

String handle.

Warning

It exits the application if called when no memory space is available.

Examples:

[example_httpuids.c](#), [example_router_srv.c](#), and [example_str.c](#).

5.3.2.2 [sg_str_free\(\)](#)

```
void sg\_str\_free (  
    struct sg\_str * str )
```

Frees the string handle previously allocated by [sg_str_new\(\)](#).

Parameters

in	<i>str</i>	Pointer of the string handle to be freed.
----	------------	---

Examples:

[example_httpuplds.c](#), [example_router_srv.c](#), and [example_str.c](#).

5.3.2.3 sg_str_write()

```
int sg_str_write (
    struct sg_str * str,
    const char * val,
    size_t len )
```

Writes a null-terminated string to the string handle ***str***. All strings previously written are kept.

Parameters

in	<i>str</i>	String handle.
in	<i>val</i>	String to be written.
in	<i>len</i>	Length of the string to be written.

Return values

0	- Success.
<i>EINVAL</i>	- Invalid argument.

5.3.2.4 sg_str_printf_va()

```
int sg_str_printf_va (
    struct sg_str * str,
    const char * fmt,
    va_list ap )
```

Prints a null-terminated formatted string from the argument list to the string handle ***str***.

Parameters

in	<i>str</i>	String handle.
in	<i>fmt</i>	Formatted string (following the same <code>printf()</code> format specification).
in	<i>ap</i>	Arguments list (handled by <code>va_start()</code> / <code>va_end()</code>).

Return values

0	- Success.
---	------------

Return values

<i>EINVAL</i>	- Invalid argument.
---------------	---------------------

5.3.2.5 sg_str_printf()

```
int sg_str_printf (
    struct sg_str * str,
    const char * fmt,
    ... )
```

Prints a null-terminated formatted string to the string handle **str**. All strings previously written are kept.

Parameters

in	<i>str</i>	String handle.
in	<i>fmt</i>	Formatted string (following the same <code>printf()</code> format specification).
in	...	Additional arguments (following the same <code>printf()</code> arguments specification).

Return values

0	- Success.
<i>EINVAL</i>	- Invalid argument.

Examples:

[example_httpuids.c](#), [example_router_srv.c](#), and [example_str.c](#).

5.3.2.6 sg_str_content()

```
int const char* sg_str_content (
    struct sg_str * str )
```

Returns the null-terminated string content from the string handle **str**.

Parameters

in	<i>str</i>	String handle.
----	------------	----------------

Returns

Content as null-terminated string.

Return values

<i>NULL</i>	If the str is null and sets the <code>errno</code> to <code>EINVAL</code> .
-------------	--

Examples:

[example_httpuplds.c](#), [example_router_srv.c](#), and [example_str.c](#).

5.3.2.7 `sg_str_length()`

```
size_t sg_str_length (
    struct sg_str * str )
```

Returns the total string length from the handle **str**.

Parameters

in	<i>str</i>	String handle.
----	------------	----------------

Returns

Total string length.

Return values

<i>EINVAL</i>	- Invalid argument.
---------------	---------------------

5.3.2.8 `sg_str_clear()`

```
int sg_str_clear (
    struct sg_str * str )
```

Cleans all existing content in the string handle **str**.

Parameters

in	<i>str</i>	String handle.
----	------------	----------------

Return values

<i>0</i>	- Success.
<i>EINVAL</i>	- Invalid argument.

Examples:

[example_httpuplds.c](#).

5.4 String map

Data Structures

- struct [sg_strmap](#)

Typedefs

- typedef int(* [sg_strmap_iter_cb](#)) (void *cls, struct [sg_strmap](#) *pair)
- typedef int(* [sg_strmap_sort_cb](#)) (void *cls, struct [sg_strmap](#) *pair_a, struct [sg_strmap](#) *pair_b)

Functions

- const char * [sg_strmap_name](#) (struct [sg_strmap](#) *pair)
- const char * [sg_strmap_val](#) (struct [sg_strmap](#) *pair)
- int [sg_strmap_add](#) (struct [sg_strmap](#) **map, const char *name, const char *val)
- int [sg_strmap_set](#) (struct [sg_strmap](#) **map, const char *name, const char *val)
- int [sg_strmap_find](#) (struct [sg_strmap](#) *map, const char *name, struct [sg_strmap](#) **pair)
- const char * [sg_strmap_get](#) (struct [sg_strmap](#) *map, const char *name)
- int [sg_strmap_rm](#) (struct [sg_strmap](#) **map, const char *name)
- int [sg_strmap_iter](#) (struct [sg_strmap](#) *map, [sg_strmap_iter_cb](#) cb, void *cls)
- int [sg_strmap_sort](#) (struct [sg_strmap](#) **map, [sg_strmap_sort_cb](#) cb, void *cls)
- unsigned int [sg_strmap_count](#) (struct [sg_strmap](#) *map)
- int [sg_strmap_next](#) (struct [sg_strmap](#) **next)
- void [sg_strmap_cleanup](#) (struct [sg_strmap](#) **map)

5.4.1 Detailed Description

String map handle and its related functions.

5.4.2 Typedef Documentation

5.4.2.1 [sg_strmap_iter_cb](#)

```
typedef int(* sg_strmap_iter_cb) (void *cls, struct sg\_strmap *pair)
```

Callback signature used by [sg_strmap_iter\(\)](#) to iterate pairs of strings.

Parameters

out	<i>cls</i>	User-defined closure.
out	<i>pair</i>	Current iterated pair.

Return values

0	- Success.
---	------------

Return values

<i>E<ERROR></i>	- User-defined error to stop pairs iteration.
-----------------------	---

5.4.2.2 sg_strmap_sort_cb

```
typedef int(* sg_strmap_sort_cb) (void *cls, struct sg_strmap *pair_a, struct sg_strmap *pair↔
_b)
```

Callback signature used by [sg_strmap_sort\(\)](#) to sort pairs of strings.

Parameters

out	<i>cls</i>	User-defined closure.
out	<i>pair↔ _a</i>	Current left pair (A).
out	<i>pair↔ _b</i>	Current right pair (B).

Return values

-1	- A < B.
0	- A == B.
1	- A > B.

5.4.3 Function Documentation

5.4.3.1 sg_strmap_name()

```
const char* sg_strmap_name (
    struct sg_strmap * pair )
```

Returns a name from the **pair**.

Parameters

in	<i>pair</i>	Pair of name-value.
----	-------------	---------------------

Returns

Name as null-terminated string.

Return values

<i>NULL</i>	If the pair is null and sets the <code>errno</code> to <code>EINVAL</code> .
-------------	---

Examples:

[example_strmap.c](#).

5.4.3.2 sg_strmap_val()

```
const char* sg_strmap_val (
    struct sg_strmap * pair )
```

Returns a value from the **pair**.

Parameters

in	<i>pair</i>	Pair of name-value.
----	-------------	---------------------

Returns

Value as null-terminated string.

Return values

<i>NULL</i>	If the pair is null and sets the errno to EINVAL .
-------------	---

Examples:

[example_strmap.c](#).

5.4.3.3 sg_strmap_add()

```
int sg_strmap_add (
    struct sg_strmap ** map,
    const char * name,
    const char * val )
```

Adds a pair of name-value to the string **map**.

Parameters

in, out	<i>map</i>	Pairs map pointer to add a new pair.
in	<i>name</i>	Pair name.
in	<i>val</i>	Pair value.

Return values

<i>0</i>	- Success.
<i>EINVAL</i>	- Invalid argument.

Note

It cannot check if a name already exists in a pair added to the **map**, then the uniqueness must be managed by the application.

Warning

It exits the application if called when no memory space is available.

5.4.3.4 sg_strmap_set()

```
int sg_strmap_set (
    struct sg_strmap ** map,
    const char * name,
    const char * val )
```

Sets a pair of name-value to the string **map**.

Parameters

in, out	<i>map</i>	Pairs map pointer to set a new pair.
in	<i>name</i>	Pair name.
in	<i>val</i>	Pair value.

Return values

0	- Success.
EINVAL	- Invalid argument.

Note

If a name already exists in a pair previously added into the **map**, then the function replaces its value, otherwise it is added as a new pair.

Warning

It exits the application if called when no memory space is available.

Examples:

[example_strmap.c](#).

5.4.3.5 sg_strmap_find()

```
int sg_strmap_find (
    struct sg_strmap * map,
    const char * name,
    struct sg_strmap ** pair )
```

Finds a pair by name.

Parameters

in	<i>map</i>	Pairs map.
in	<i>name</i>	Name to find the pair.
in, out	<i>pair</i>	Pointer of the variable to store the found pair.

Return values

0	- Success.
<i>EINVAL</i>	- Invalid argument.
<i>ENOENT</i>	- Pair not found.

Examples:

[example_strmap.c](#).

5.4.3.6 sg_strmap_get()

```
const char* sg_strmap_get (
    struct sg_strmap * map,
    const char * name )
```

Gets a pair by name and returns the value.

Parameters

in	<i>map</i>	Pairs map.
in	<i>name</i>	Name to get the pair.

Returns

Pair value as null-terminated string.

Return values

<i>NULL</i>	If map or name is null or pair is not found.
-------------	--

Examples:

[example_httpcookie.c](#), and [example_httpuplds.c](#).

5.4.3.7 sg_strmap_rm()

```
int sg_strmap_rm (
    struct sg_strmap ** map,
    const char * name )
```

Removes a pair by name.

Parameters

in	<i>map</i>	Pointer to the pairs map.
in	<i>name</i>	Name to find and then remove the pair.

Return values

0	- Success.
<i>EINVAL</i>	- Invalid argument.
<i>ENOENT</i>	- Pair already removed.

5.4.3.8 sg_strmap_iter()

```
int sg_strmap_iter (
    struct sg_strmap * map,
    sg_strmap_iter_cb cb,
    void * cls )
```

Iterates over pairs map.

Parameters

in	<i>map</i>	Pairs map.
in	<i>cb</i>	Callback to iterate the pairs.
in, out	<i>cls</i>	User-specified value.

Return values

0	- Success.
<i>EINVAL</i>	- Invalid argument.

Returns

Callback result when it is different from 0.

Examples:

[example_strmap.c](#).

5.4.3.9 sg_strmap_sort()

```
int sg_strmap_sort (
    struct sg_strmap ** map,
    sg_strmap_sort_cb cb,
    void * cls )
```

Sorts the pairs map.

Parameters

<code>in, out</code>	<code>map</code>	Pointer to the pairs map.
<code>in</code>	<code>cb</code>	Callback to sort the pairs.
<code>in, out</code>	<code>cls</code>	User-specified value.

Return values

<code>0</code>	- Success.
<code>EINVAL</code>	- Invalid argument.

Examples:

[example_strmap.c](#).

5.4.3.10 sg_strmap_count()

```
unsigned int sg_strmap_count (
    struct sg_strmap * map )
```

Counts the total pairs in the map.

Parameters

<code>in</code>	<code>map</code>	Pairs map.
-----------------	------------------	------------

Returns

Total of pairs.

Return values

<code>0</code>	If the list is empty or null.
----------------	-------------------------------

5.4.3.11 sg_strmap_next()

```
int sg_strmap_next (
    struct sg_strmap ** next )
```

Returns the next pair in the map.

Parameters

<code>in, out</code>	<code>next</code>	Pointer to the next pair.
----------------------	-------------------	---------------------------

Return values

<i>0</i>	- Success.
<i>EINVAL</i>	- Invalid argument.

5.4.3.12 `sg_strmap_cleanup()`

```
void sg_strmap_cleanup (
    struct sg\_strmap ** map )
```

Cleans the entire map.

Parameters

<i>in, out</i>	<i>map</i>	Pointer to the pairs map.
----------------	------------	---------------------------

Examples:

[example_strmap.c](#).

5.5 HTTP server

Data Structures

- struct [sg_httpauth](#)
- struct [sg_httpupld](#)
- struct [sg_httpreq](#)
- struct [sg_httpres](#)
- struct [sg_httpsrv](#)

Macros

- #define [sg_httpres_send](#)(res, val, content_type, status) [sg_httpres_sendbinary](#)((res), (void *) (val), ((val != NULL) ? strlen((val)) : 0), (content_type), (status))

Typedefs

- typedef bool(* [sg_httpauth_cb](#)) (void *cls, struct [sg_httpauth](#) *auth, struct [sg_httpreq](#) *req, struct [sg_httpres](#) *res)
- typedef int(* [sg_httpupld_cb](#)) (void *cls, void **handle, const char *dir, const char *field, const char *name, const char *mime, const char *encoding)
- typedef int(* [sg_httpuplds_iter_cb](#)) (void *cls, struct [sg_httpupld](#) *upld)
- typedef void(* [sg_httpreq_cb](#)) (void *cls, struct [sg_httpreq](#) *req, struct [sg_httpres](#) *res)

Functions

- int [sg_httpauth_set_realm](#) (struct [sg_httpauth](#) *auth, const char *realm)
- const char * [sg_httpauth_realm](#) (struct [sg_httpauth](#) *auth)
- int [sg_httpauth_deny](#) (struct [sg_httpauth](#) *auth, const char *justification, const char *content_type)
- int [sg_httpauth_cancel](#) (struct [sg_httpauth](#) *auth)
- const char * [sg_httpauth_usr](#) (struct [sg_httpauth](#) *auth)
- const char * [sg_httpauth_pwd](#) (struct [sg_httpauth](#) *auth)
- int [sg_httpuplds_iter](#) (struct [sg_httpupld](#) *uplds, [sg_httpuplds_iter_cb](#) cb, void *cls)
- int [sg_httpuplds_next](#) (struct [sg_httpupld](#) **upld)
- unsigned int [sg_httpuplds_count](#) (struct [sg_httpupld](#) *uplds)
- void * [sg_httpupld_handle](#) (struct [sg_httpupld](#) *upld)
- const char * [sg_httpupld_dir](#) (struct [sg_httpupld](#) *upld)
- const char * [sg_httpupld_field](#) (struct [sg_httpupld](#) *upld)
- const char * [sg_httpupld_name](#) (struct [sg_httpupld](#) *upld)
- const char * [sg_httpupld_mime](#) (struct [sg_httpupld](#) *upld)
- const char * [sg_httpupld_encoding](#) (struct [sg_httpupld](#) *upld)
- uint64_t [sg_httpupld_size](#) (struct [sg_httpupld](#) *upld)
- int [sg_httpupld_save](#) (struct [sg_httpupld](#) *upld, bool overwritten)
- int [sg_httpupld_save_as](#) (struct [sg_httpupld](#) *upld, const char *path, bool overwritten)
- struct [sg_strmap](#) ** [sg_httpreq_headers](#) (struct [sg_httpreq](#) *req)
- struct [sg_strmap](#) ** [sg_httpreq_cookies](#) (struct [sg_httpreq](#) *req)
- struct [sg_strmap](#) ** [sg_httpreq_params](#) (struct [sg_httpreq](#) *req)
- struct [sg_strmap](#) ** [sg_httpreq_fields](#) (struct [sg_httpreq](#) *req)
- const char * [sg_httpreq_version](#) (struct [sg_httpreq](#) *req)
- const char * [sg_httpreq_method](#) (struct [sg_httpreq](#) *req)
- const char * [sg_httpreq_path](#) (struct [sg_httpreq](#) *req)
- struct [sg_str](#) * [sg_httpreq_payload](#) (struct [sg_httpreq](#) *req)

- bool `sg_httpreq_is_uploading` (struct `sg_httpreq` *req)
- struct `sg_httpupld` * `sg_httpreq_uploads` (struct `sg_httpreq` *req)
- void * `sg_httpreq_tls_session` (struct `sg_httpreq` *req)
- int `sg_httpreq_set_user_data` (struct `sg_httpreq` *req, void *data)
- void * `sg_httpreq_user_data` (struct `sg_httpreq` *req)
- struct `sg_strmap` ** `sg_httpres_headers` (struct `sg_httpres` *res)
- int `sg_httpres_set_cookie` (struct `sg_httpres` *res, const char *name, const char *val)
- int `sg_httpres_sendbinary` (struct `sg_httpres` *res, void *buf, size_t size, const char *content_type, unsigned int status)
- int `sg_httpres_sendfile` (struct `sg_httpres` *res, size_t block_size, uint64_t max_size, const char *filename, bool rendered, unsigned int status)
- int `sg_httpres_sendstream` (struct `sg_httpres` *res, uint64_t size, size_t block_size, `sg_read_cb` read_cb, void *handle, `sg_free_cb` free_cb, unsigned int status)
- int `sg_httpres_clear` (struct `sg_httpres` *res)
- struct `sg_httpsrv` * `sg_httpsrv_new2` (`sg_httpauth_cb` auth_cb, void *auth_cls, `sg_httpreq_cb` req_cb, void *req_cls, `sg_err_cb` err_cb, void *err_cls) `__attribute__((malloc))`
- struct `sg_httpsrv` * `sg_httpsrv_new` (`sg_httpreq_cb` cb, void *cls) `__attribute__((malloc))`
- void `sg_httpsrv_free` (struct `sg_httpsrv` *srv)
- bool `sg_httpsrv_tls_listen2` (struct `sg_httpsrv` *srv, const char *key, const char *pwd, const char *cert, const char *trust, const char *dhparams, uint16_t port, bool threaded)
- bool `sg_httpsrv_tls_listen` (struct `sg_httpsrv` *srv, const char *key, const char *cert, uint16_t port, bool threaded)
- bool `sg_httpsrv_listen` (struct `sg_httpsrv` *srv, uint16_t port, bool threaded)
- int `sg_httpsrv_shutdown` (struct `sg_httpsrv` *srv)
- uint16_t `sg_httpsrv_port` (struct `sg_httpsrv` *srv)
- bool `sg_httpsrv_is_threaded` (struct `sg_httpsrv` *srv)
- int `sg_httpsrv_set_upld_cbs` (struct `sg_httpsrv` *srv, `sg_httpupld_cb` cb, void *cls, `sg_write_cb` write_cb, `sg_free_cb` free_cb, `sg_save_cb` save_cb, `sg_save_as_cb` save_as_cb)
- int `sg_httpsrv_set_upld_dir` (struct `sg_httpsrv` *srv, const char *dir)
- const char * `sg_httpsrv_upld_dir` (struct `sg_httpsrv` *srv)
- int `sg_httpsrv_set_post_buf_size` (struct `sg_httpsrv` *srv, size_t size)
- size_t `sg_httpsrv_post_buf_size` (struct `sg_httpsrv` *srv)
- int `sg_httpsrv_set_payld_limit` (struct `sg_httpsrv` *srv, size_t limit)
- size_t `sg_httpsrv_payld_limit` (struct `sg_httpsrv` *srv)
- int `sg_httpsrv_set_uplds_limit` (struct `sg_httpsrv` *srv, uint64_t limit)
- uint64_t `sg_httpsrv_uplds_limit` (struct `sg_httpsrv` *srv)
- int `sg_httpsrv_set_thr_pool_size` (struct `sg_httpsrv` *srv, unsigned int size)
- unsigned int `sg_httpsrv_thr_pool_size` (struct `sg_httpsrv` *srv)
- int `sg_httpsrv_set_con_timeout` (struct `sg_httpsrv` *srv, unsigned int timeout)
- unsigned int `sg_httpsrv_con_timeout` (struct `sg_httpsrv` *srv)
- int `sg_httpsrv_set_con_limit` (struct `sg_httpsrv` *srv, unsigned int limit)
- unsigned int `sg_httpsrv_con_limit` (struct `sg_httpsrv` *srv)
- ssize_t `sg_httpread_end` (bool err)

5.5.1 Detailed Description

Fast event-driven HTTP server.

5.5.2 Macro Definition Documentation

5.5.2.1 sg_httpres_send

```
#define sg_httpres_send(
    res,
    val,
    content_type,
    status ) sg_httpres_sendbinary((res), (void *) (val), ((val != NULL) ? strlen((val))
: 0), (content_type), (status))
```

Sends a null-terminated string content to the client.

Parameters

in	<i>res</i>	Response handle.
in	<i>val</i>	Null-terminated string.
in	<i>content_type</i>	Content-Type of the content.
in	<i>status</i>	HTTP status code.

Return values

0	- Success.
<i>EINVAL</i>	- Invalid argument.
<i>EALREADY</i>	- Operation already in progress.

Warning

It exits the application if called when no memory space is available.

Examples:

[example_httpauth.c](#), [example_httpcookie.c](#), [example_httpsrv.c](#), [example_httpsrv_benchmark.c](#), [example_httpsrv_tls.c](#), [example_httpsrv_tls_cert_auth.c](#), [example_httpuplds.c](#), and [example_router_srv.c](#).

5.5.3 Typedef Documentation

5.5.3.1 sg_httpauth_cb

```
typedef bool(* sg_httpauth_cb) (void *cls, struct sg_httpauth *auth, struct sg_httpreq *req,
struct sg_httpres *res)
```

Callback signature used to grant or deny the user access to the server resources.

Parameters

out	<i>cls</i>	User-defined closure.
out	<i>auth</i>	Authentication handle.
out	<i>req</i>	Request handle.
out	<i>res</i>	Response handle.

Return values

<i>true</i>	Grants the user access.
<i>false</i>	Denies the user access.

5.5.3.2 sg_httupld_cb

```
typedef int(* sg_httupld_cb) (void *cls, void **handle, const char *dir, const char *field,
const char *name, const char *mime, const char *encoding)
```

Callback signature used to handle uploaded files and/or fields.

Parameters

out	<i>cls</i>	User-defined closure.
in, out	<i>handle</i>	Stream handle pointer.
out	<i>dir</i>	Directory to store the uploaded files.
out	<i>field</i>	Posted field.
out	<i>name</i>	Uploaded file name.
out	<i>mime</i>	Uploaded file content-type (e.g.: text/plain, image/png, application/json etc.).
out	<i>encoding</i>	Uploaded file transfer-encoding (e.g.: chunked, deflate, gzip etc.).

Return values

<i>0</i>	- Success.
<i>E<ERROR></i>	- User-defined error to refuse the upload.

5.5.3.3 sg_httpuplds_iter_cb

```
typedef int(* sg_httpuplds_iter_cb) (void *cls, struct sg_httpupld *upld)
```

Callback signature used to iterate uploaded files.

Parameters

out	<i>cls</i>	User-defined closure.
out	<i>upld</i>	Current upload item.

Return values

<i>0</i>	- Success.
<i>E<ERROR></i>	- User-defined error to stop list iteration.

5.5.3.4 sg_httpreq_cb

```
typedef void(* sg_httpreq_cb) (void *cls, struct sg_httpreq *req, struct sg_httpres *res)
```

Callback signature used to handle requests and responses.

Parameters

out	<i>cls</i>	User-defined closure.
out	<i>req</i>	Request handle.
out	<i>res</i>	Response handle.

5.5.4 Function Documentation

5.5.4.1 sg_httpauth_set_realm()

```
int sg_httpauth_set_realm (
    struct sg_httpauth * auth,
    const char * realm )
```

Sets the authentication protection space (realm).

Parameters

in	<i>auth</i>	Authentication handle.
in	<i>realm</i>	Realm string.

Return values

<i>0</i>	- Success.
<i>EINVAL</i>	- Invalid argument.
<i>EALREADY</i>	- Realm already set.

Warning

It exits the application if called when no memory space is available.

Examples:

[example_httpauth.c](#).

5.5.4.2 sg_httpauth_realm()

```
const char* sg_httpauth_realm (
    struct sg_httpauth * auth )
```

Gets the authentication protection space (realm).

Parameters

in	<i>auth</i>	Authentication handle.
----	-------------	------------------------

Returns

Realm as null-terminated string.

Return values

<i>NULL</i>	If <i>auth</i> is null and sets the <code>errno</code> to <code>EINVAL</code> .
-------------	---

5.5.4.3 `sg_httpauth_deny()`

```
int sg_httpauth_deny (
    struct sg_httpauth * auth,
    const char * justification,
    const char * content_type )
```

Deny the authentication sending a justification to the user.

Parameters

in	<i>auth</i>	Authentication handle.
in	<i>justification</i>	Justification message.
in	<i>content_type</i>	Content-Type of the justification.

Return values

<i>0</i>	- Success.
<i>EINVAL</i>	- Invalid argument.
<i>EALREADY</i>	- Already denied.

Examples:

[example_httpauth.c](#).

5.5.4.4 `sg_httpauth_cancel()`

```
int sg_httpauth_cancel (
    struct sg_httpauth * auth )
```

Cancels the authentication loop while the user is trying to access the server.

Parameters

in	<i>auth</i>	Authentication handle.
----	-------------	------------------------

Return values

0	- Success.
<i>EINVAL</i>	- Invalid argument.

5.5.4.5 sg_httpauth_usr()

```
const char* sg_httpauth_usr (  
    struct sg_httpauth * auth )
```

Returns the authentication user.

Parameters

in	<i>auth</i>	Authentication handle.
----	-------------	------------------------

Returns

User as null-terminated string.

Return values

<i>NULL</i>	If auth is null and sets the <code>errno</code> to <code>EINVAL</code> .
-------------	---

Examples:

[example_httpauth.c](#).

5.5.4.6 sg_httpauth_pwd()

```
const char* sg_httpauth_pwd (  
    struct sg_httpauth * auth )
```

Returns the authentication password.

Parameters

in	<i>auth</i>	Authentication handle.
----	-------------	------------------------

Returns

Password as null-terminated string.

Return values

<i>NULL</i>	If auth is null and sets the <code>errno</code> to <code>EINVAL</code> .
-------------	---

Examples:

[example_httpauth.c](#).

5.5.4.7 sg_httpuplds_iter()

```
int sg_httpuplds_iter (
    struct sg_httpupld * uplds,
    sg_httpuplds_iter_cb cb,
    void * cls )
```

Iterates over all the upload items in the **uplds** list.

Parameters

in	<i>uplds</i>	Uploads list handle.
in	<i>cb</i>	Callback to iterate over upload items.
in	<i>cls</i>	User-defined closure.

Return values

<i>0</i>	- Success.
<i>EINVAL</i>	- Invalid argument.
<i>E<ERROR></i>	- User-defined error to abort the list iteration.

5.5.4.8 sg_httpuplds_next()

```
int sg_httpuplds_next (
    struct sg_httpupld ** upld )
```

Gets the next upload item starting from the first item pointer **upld**.

Parameters

in, out	<i>upld</i>	Next upload item starting from the first item pointer.
---------	-------------	--

Return values

0	- Success.
<i>EINVAL</i>	- Invalid argument.

Examples:

[example_httpuplds.c](#).

5.5.4.9 sg_httpuplds_count()

```
unsigned int sg_httpuplds_count (
    struct sg_httpupld * uplds )
```

Counts the total upload items in the list **uplds**.

Parameters

in	<i>uplds</i>	Uploads list.
----	--------------	---------------

Returns

Total of items.

Return values

0	If the list is empty or null.
---	-------------------------------

5.5.4.10 sg_httpupld_handle()

```
void* sg_httpupld_handle (
    struct sg_httpupld * upld )
```

Returns the stream handle of the upload handle **upld**.

Parameters

in	<i>upld</i>	Upload handle.
----	-------------	----------------

Returns

Stream handle.

Return values

<code>NULL</code>	If <code>upld</code> is null and sets the <code>errno</code> to <code>EINVAL</code> .
-------------------	---

5.5.4.11 `sg_httpupld_dir()`

```
const char* sg_httpupld_dir (  
    struct sg_httpupld * upld )
```

Returns the directory of the upload handle `upld`.

Parameters

in	<code>upld</code>	Upload handle.
----	-------------------	----------------

Returns

Upload directory as null-terminated string.

Return values

<code>NULL</code>	If <code>upld</code> is null and sets the <code>errno</code> to <code>EINVAL</code> .
-------------------	---

5.5.4.12 `sg_httpupld_field()`

```
const char* sg_httpupld_field (  
    struct sg_httpupld * upld )
```

Returns the field of the upload handle `upld`.

Parameters

in	<code>upld</code>	Upload handle.
----	-------------------	----------------

Returns

Upload field as null-terminated string.

Return values

<code>NULL</code>	If <code>upld</code> is null and sets the <code>errno</code> to <code>EINVAL</code> .
-------------------	---

5.5.4.13 sg_httpupld_name()

```
const char* sg_httpupld_name (  
    struct sg_httpupld * upld )
```

Returns the name of the upload handle **upld**.

Parameters

in	<i>upld</i>	Upload handle.
----	-------------	----------------

Returns

Upload name as null-terminated string.

Return values

NULL	If upld is null and sets the <code>errno</code> to <code>EINVAL</code> .
-------------	---

Examples:

[example_httpuplds.c](#).

5.5.4.14 sg_httpupld_mime()

```
const char* sg_httpupld_mime (  
    struct sg_httpupld * upld )
```

Returns the MIME (content-type) of the upload.

Parameters

in	<i>upld</i>	Upload handle.
----	-------------	----------------

Returns

Upload MIME as null-terminated string.

Return values

NULL	If upld is null and sets the <code>errno</code> to <code>EINVAL</code> .
-------------	---

5.5.4.15 sg_httpupld_encoding()

```
const char* sg_httpupld_encoding (  
    struct sg_httpupld * upld )
```

Returns the encoding (transfer-encoding) of the upload.

Parameters

in	<i>upld</i>	Upload handle.
----	-------------	----------------

Returns

Upload encoding as null-terminated string.

Return values

<i>NULL</i>	If upld is null and sets the <code>errno</code> to <code>EINVAL</code> .
-------------	---

5.5.4.16 `sg_httpupld_size()`

```
uint64_t sg_httpupld_size (
    struct sg_httpupld * upld )
```

Returns the size of the upload.

Parameters

in	<i>upld</i>	Upload handle.
----	-------------	----------------

Returns

Upload size into `uint64`. If **upld** is null, sets the `errno` to `EINVAL`.

5.5.4.17 `sg_httpupld_save()`

```
int sg_httpupld_save (
    struct sg_httpupld * upld,
    bool overwritten )
```

Saves the uploaded file defining the destination path by upload name and directory.

Parameters

in	<i>upld</i>	Upload handle.
in	<i>overwritten</i>	Overwrite upload file if it exists.

Return values

0	- Success.
---	------------

Return values

<i>EINVAL</i>	- Invalid argument.
<i>EEXIST</i>	- File already exists (if overwritten is <i>false</i>).
<i>EISDIR</i>	- Destination file is a directory.

Examples:

[example_httuplds.c](#).

5.5.4.18 `sg_httupld_save_as()`

```
int sg_httupld_save_as (
    struct sg_httupld * upld,
    const char * path,
    bool overwritten )
```

Saves the uploaded file allowing to define the destination path.

Parameters

in	<i>upld</i>	Upload handle.
in	<i>path</i>	Absolute destination path.
in	<i>overwritten</i>	Overwrite upload file if it exists.

Return values

<i>0</i>	- Success.
<i>EINVAL</i>	- Invalid argument.
<i>EEXIST</i>	- File already exists (if overwritten is <i>true</i>).
<i>EISDIR</i>	- Destination file is a directory.

5.5.4.19 `sg_httpreq_headers()`

```
struct sg_strmap** sg_httpreq_headers (
    struct sg_httpreq * req )
```

Returns the client headers into `sg_strmap` map.

Parameters

in	<i>req</i>	Request handle.
----	------------	-----------------

Returns

Reference to the client headers map.

Return values

<code>NULL</code>	If <code>req</code> is null and sets the <code>errno</code> to <code>EINVAL</code>
-------------------	--

Note

The headers map is automatically freed by the library.

5.5.4.20 sg_httpreq_cookies()

```
struct sg_strmap** sg_httpreq_cookies (  
    struct sg_httpreq * req )
```

Returns the client cookies into `sg_strmap` map.

Parameters

in	<code>req</code>	Request handle.
----	------------------	-----------------

Returns

Reference to the client cookies map.

Return values

<code>NULL</code>	If <code>req</code> is null and sets the <code>errno</code> to <code>EINVAL</code>
-------------------	--

Note

The cookies map is automatically freed by the library.

Examples:

[example_httpcookie.c](#).

5.5.4.21 sg_httpreq_params()

```
struct sg_strmap** sg_httpreq_params (  
    struct sg_httpreq * req )
```

Returns the query-string into `sg_strmap` map.

Parameters

in	<i>req</i>	Request handle.
----	------------	-----------------

Returns

Reference to the query-string map.

Return values

<i>NULL</i>	If req is null and sets the <i>errno</i> to <i>EINVAL</i>
-------------	--

Note

The query-string map is automatically freed by the library.

Examples:

[example_httpuplds.c](#).

5.5.4.22 sg_httpreq_fields()

```
struct sg_strmap** sg_httpreq_fields (
    struct sg_httpreq * req )
```

Returns the fields of a HTML form into *sg_strmap* map.

Parameters

in	<i>req</i>	Request handle.
----	------------	-----------------

Returns

Reference to the form fields map.

Return values

<i>NULL</i>	If req is null and sets the <i>errno</i> to <i>EINVAL</i>
-------------	--

Note

The form fields map is automatically freed by the library.

5.5.4.23 sg_httpreq_version()

```
const char* sg_httpreq_version (  
    struct sg_httpreq * req )
```

Returns the HTTP version.

Parameters

in	<i>req</i>	Request handle.
----	------------	-----------------

Returns

HTTP version as null-terminated string.

Return values

<i>NULL</i>	If req is null and sets the <code>errno</code> to <code>EINVAL</code> .
-------------	--

5.5.4.24 sg_httpreq_method()

```
const char* sg_httpreq_method (  
    struct sg_httpreq * req )
```

Returns the HTTP method.

Parameters

in	<i>req</i>	Request handle.
----	------------	-----------------

Returns

HTTP method as null-terminated string.

Return values

<i>NULL</i>	If req is null and sets the <code>errno</code> to <code>EINVAL</code> .
-------------	--

5.5.4.25 sg_httpreq_path()

```
const char* sg_httpreq_path (  
    struct sg_httpreq * req )
```

Returns the path component.

Parameters

in	<i>req</i>	Request handle.
----	------------	-----------------

Returns

Path component as null-terminated string.

Return values

<i>NULL</i>	If <i>req</i> is null and sets the <i>errno</i> to <i>EINVAL</i> .
-------------	---

Examples:

[example_httpcookie.c](#), and [example_router_srv.c](#).

5.5.4.26 sg_httpreq_payload()

```
struct sg_str* sg_httpreq_payload (  
    struct sg_httpreq * req )
```

Returns the posting payload into a [sg_str](#) instance.

Parameters

in	<i>req</i>	Request handle.
----	------------	-----------------

Returns

Instance of the payload.

Return values

<i>NULL</i>	If <i>req</i> is null and sets the <i>errno</i> to <i>EINVAL</i> .
-------------	---

Note

The form payload instance is automatically freed by the library.

5.5.4.27 sg_httpreq_is_uploading()

```
bool sg_httpreq_is_uploading (  
    struct sg_httpreq * req )
```

Checks if the client is uploading data.

Parameters

in	<i>req</i>	Request handle.
----	------------	-----------------

Returns

`true` if the client is uploading data, `false` otherwise. If ***req*** is null, sets the `errno` to `EINVAL`.

Examples:

[example_httpuplds.c](#).

5.5.4.28 `sg_httpreq_uploads()`

```
struct sg_httpupld* sg_httpreq_uploads (
    struct sg_httpreq * req )
```

Returns the list of the uploaded files.

Parameters

in	<i>req</i>	Request handle.
----	------------	-----------------

Returns

List of the uploaded files.

Return values

<code>NULL</code>	If <i>req</i> is null and sets the <code>errno</code> to <code>EINVAL</code> .
-------------------	---

Note

The uploads list is automatically freed by the library.

Examples:

[example_httpuplds.c](#).

5.5.4.29 `sg_httpreq_tls_session()`

```
void* sg_httpreq_tls_session (
    struct sg_httpreq * req )
```

Returns the GnuTLS session handle.

Parameters

in	<i>req</i>	Request handle.
----	------------	-----------------

Return values

0	- Success.
<i>EINVAL</i>	- Invalid argument.

Examples:

[example_httpsrv_tls_cert_auth.c](#).

5.5.4.30 sg_httpreq_set_user_data()

```
int sg_httpreq_set_user_data (
    struct sg_httpreq * req,
    void * data )
```

Sets user data to the request handle.

Parameters

in	<i>req</i>	Request handle.
in	<i>data</i>	User data pointer.

Return values

0	- Success.
<i>EINVAL</i>	- Invalid argument.

5.5.4.31 sg_httpreq_user_data()

```
void* sg_httpreq_user_data (
    struct sg_httpreq * req )
```

Gets user data from the request handle.

Parameters

in	<i>req</i>	Request handle.
----	------------	-----------------

Returns

User data pointer.

Return values

<i>NULL</i>	If req is null and sets the <code>errno</code> to <code>EINVAL</code> .
-------------	--

5.5.4.32 sg_httpres_headers()

```
struct sg_strmap** sg_httpres_headers (
    struct sg_httpres * res )
```

Returns the server headers into `sg_strmap` map.

Parameters

in	<i>res</i>	Response handle.
----	------------	------------------

Returns

Reference to the server headers map.

Return values

<i>NULL</i>	If res is null and sets the <code>errno</code> to <code>EINVAL</code>
-------------	--

Note

The headers map is automatically freed by the library.

5.5.4.33 sg_httpres_set_cookie()

```
int sg_httpres_set_cookie (
    struct sg_httpres * res,
    const char * name,
    const char * val )
```

Sets server cookie to the response handle.

Parameters

in	<i>res</i>	Response handle.
in	<i>name</i>	Cookie name.
in	<i>val</i>	Cookie value.

Return values

<i>0</i>	- Success.
<i>EINVAL</i>	- Invalid argument.

Warning

It exits the application if called when no memory space is available.

Examples:

[example_httpcookie.c](#).

5.5.4.34 sg_httpres_sendbinary()

```
int sg_httpres_sendbinary (
    struct sg_httpres * res,
    void * buf,
    size_t size,
    const char * content_type,
    unsigned int status )
```

Sends a binary content to the client.

Parameters

in	<i>res</i>	Response handle.
in	<i>buf</i>	Binary content.
in	<i>size</i>	Content size.
in	<i>content_type</i>	Content-Type of the content.
in	<i>status</i>	HTTP status code.

Return values

<i>0</i>	- Success.
<i>EINVAL</i>	- Invalid argument.
<i>EALREADY</i>	- Operation already in progress.

Warning

It exits the application if called when no memory space is available.

5.5.4.35 sg_httpres_sendfile()

```
int sg_httpres_sendfile (
    struct sg_httpres * res,
```

```

size_t block_size,
uint64_t max_size,
const char * filename,
bool rendered,
unsigned int status )

```

Sends a file to the client.

Parameters

in	<i>res</i>	Response handle.
in	<i>block_size</i>	Preferred block size for file loading.
in	<i>max_size</i>	Maximum allowed file size.
in	<i>filename</i>	Path of the file to be sent.
in	<i>rendered</i>	If <code>true</code> the file is rendered, otherwise downloaded.
in	<i>status</i>	HTTP status code.

Return values

<i>0</i>	- Success.
<i>EINVAL</i>	- Invalid argument.
<i>EALREADY</i>	- Operation already in progress.
<i>EISDIR</i>	- Is a directory.
<i>EBADF</i>	- Bad file number.
<i>EFBIG</i>	- File too large.

Warning

It exits the application if called when no memory space is available.

Examples:

[example_httpuplds.c](#).

5.5.4.36 sg_httpres_sendstream()

```

int sg_httpres_sendstream (
    struct sg_httpres * res,
    uint64_t size,
    size_t block_size,
    sg_read_cb read_cb,
    void * handle,
    sg_free_cb free_cb,
    unsigned int status )

```

Sends a stream to the client.

Parameters

in	<i>res</i>	Response handle.
in	<i>size</i>	Size of the stream.

Parameters

in	<i>block_size</i>	Preferred block size for stream loading.
in	<i>read_cb</i>	Callback to read data from stream handle.
in	<i>handle</i>	Stream handle.
in	<i>free_cb</i>	Callback to free the stream handle.
in	<i>status</i>	HTTP status code.

Return values

0	- Success.
<i>EINVAL</i>	- Invalid argument.
<i>EALREADY</i>	- Operation already in progress.

Note

Use `size = 0` if the stream size is unknown.

Warning

It exits the application if called when no memory space is available.

5.5.4.37 sg_httpres_clear()

```
int sg_httpres_clear (
    struct sg_httpres * res )
```

Clears all headers, cookies, status and internal buffers of the response handle.

Parameters

in	<i>res</i>	Response handle.
----	------------	------------------

Return values

0	- Success.
<i>EINVAL</i>	- Invalid argument.

5.5.4.38 sg_httpsrv_new2()

```
struct sg_httpsrv* sg_httpsrv_new2 (
    sg_httpauth_cb auth_cb,
    void * auth_cls,
    sg_httpreq_cb req_cb,
    void * req_cls,
```

```
sg_err_cb err_cb,
void * err_cls )
```

Creates a new HTTP server handle.

Parameters

in	<i>auth_cb</i>	Callback to grant/deny user access to the server resources.
in	<i>auth_cls</i>	User-defined closure for auth_cb .
in	<i>req_cb</i>	Callback to handle requests and responses.
in	<i>req_cls</i>	User-defined closure for req_cb .
in	<i>err_cb</i>	Callback to handle server errors.
in	<i>err_cls</i>	User-defined closure for err_cb .

Returns

New HTTP server handle.

Return values

<i>NULL</i>	If the req_cb or err_cb is null and sets the <i>errno</i> to <i>EINVAL</i> .
-------------	--

Examples:

[example_httpauth.c](#).

5.5.4.39 sg_httpsrv_new()

```
struct sg_httpsrv* sg_httpsrv_new (
    sg_httpreq_cb cb,
    void * cls )
```

Creates a new HTTP server handle.

Parameters

in	<i>cb</i>	Callback to handle requests and responses.
in	<i>cls</i>	User-defined closure.

Returns

New HTTP server handle.

Return values

<i>NULL</i>	If the cb is null and sets the <i>errno</i> to <i>EINVAL</i> .
-------------	---

Examples:

[example_httpcookie.c](#), [example_httpsrv.c](#), [example_httpsrv_benchmark.c](#), [example_httpsrv_tls.c](#), [example_httpsrv_tls_cert_auth.c](#), [example_httpuplds.c](#), and [example_router_srv.c](#).

5.5.4.40 sg_httpsrv_free()

```
void sg_httpsrv_free (
    struct sg_httpsrv * srv )
```

Frees the server handle previously allocated by [sg_httpsrv_new\(\)](#) or [sg_httpsrv_new2\(\)](#).

Parameters

in	<i>srv</i>	Pointer of the server to be freed.
----	------------	------------------------------------

Note

If the server is running it stops before being freed.

Examples:

[example_httpauth.c](#), [example_httpcookie.c](#), [example_httpsrv.c](#), [example_httpsrv_benchmark.c](#), [example_httpsrv_tls.c](#), [example_httpsrv_tls_cert_auth.c](#), [example_httpuplds.c](#), and [example_router_srv.c](#).

5.5.4.41 sg_httpsrv_tls_listen2()

```
bool sg_httpsrv_tls_listen2 (
    struct sg_httpsrv * srv,
    const char * key,
    const char * pwd,
    const char * cert,
    const char * trust,
    const char * dhparams,
    uint16_t port,
    bool threaded )
```

Starts the HTTPS server.

Parameters

in	<i>srv</i>	Server handle.
in	<i>key</i>	Memory pointer for the private key (key.pem) to be used by the HTTPS server.
in	<i>pwd</i>	Password for the private key.
in	<i>cert</i>	Memory pointer for the certificate (cert.pem) to be used by the HTTPS server.
in	<i>trust</i>	Memory pointer for the certificate (ca.pem) to be used by the HTTPS server for client authentication.
in	<i>dhparams</i>	Memory pointer for the Diffie Hellman parameters (dh.pem) to be used by the HTTPS server for key exchange.
in	<i>port</i>	Port for listening to connections.
in	<i>threaded</i>	Enable/disable the threaded model. If <code>true</code> , the server creates one thread per connection.

Returns

`true` if the server is started, `false` otherwise. If `srv` is null, sets the `errno` to `EINVAL`.

Note

If port is 0, the operating system will assign an unused port randomly.

Examples:

[example_httpsrv_tls_cert_auth.c](#).

5.5.4.42 sg_httpsrv_tls_listen()

```
bool sg_httpsrv_tls_listen (
    struct sg_httpsrv * srv,
    const char * key,
    const char * cert,
    uint16_t port,
    bool threaded )
```

Starts the HTTPS server.

Parameters

in	<i>srv</i>	Server handle.
in	<i>key</i>	Memory pointer for the private key (key.pem) to be used by the HTTPS server.
in	<i>cert</i>	Memory pointer for the certificate (cert.pem) to be used by the HTTPS server.
in	<i>port</i>	Port for listening to connections.
in	<i>threaded</i>	Enable/disable the threaded model. If <code>true</code> , the server creates one thread per connection.

Returns

`true` if the server is started, `false` otherwise. If `srv` is null, sets the `errno` to `EINVAL`.

Note

If port is 0, the operating system will assign an unused port randomly.

Examples:

[example_httpsrv_tls.c](#).

5.5.4.43 sg_httpsrv_listen()

```
bool sg_httpsrv_listen (
    struct sg_httpsrv * srv,
    uint16_t port,
    bool threaded )
```

Starts the HTTP server.

Parameters

in	<i>srv</i>	Server handle.
in	<i>port</i>	Port for listening to connections.
in	<i>threaded</i>	Enable/disable the threaded model. If <code>true</code> , the server creates one thread per connection.

Returns

`true` if the server is started, `false` otherwise. If ***srv*** is null, sets the `errno` to `EINVAL`.

Note

If port is 0, the operating system will assign randomly an unused port.

Examples:

[example_httpauth.c](#), [example_httpcookie.c](#), [example_httpsrv.c](#), [example_httpsrv_benchmark.c](#), [example_httpuplds.c](#), and [example_router_srv.c](#).

5.5.4.44 sg_httpsrv_shutdown()

```
int sg_httpsrv_shutdown (
    struct sg_httpsrv * srv )
```

Stops the server not to accept new connections.

Parameters

in	<i>srv</i>	Server handle.
----	------------	----------------

Returns

0 if the server is stopped. If ***srv*** is null, sets the `errno` to `EINVAL`.

5.5.4.45 sg_httpsrv_port()

```
uint16_t sg_httpsrv_port (
    struct sg_httpsrv * srv )
```

Returns the server listening port.

Parameters

in	<i>srv</i>	Server handle.
----	------------	----------------

Returns

Server listening port, 0 otherwise. If **srv** is null, sets the `errno` to `EINVAL`.

Examples:

[example_httpauth.c](#), [example_httpcookie.c](#), [example_httpsrv.c](#), [example_httpsrv_benchmark.c](#), [example_httpsrv_tls.c](#), [example_httpsrv_tls_cert_auth.c](#), [example_httpuplds.c](#), and [example_router_srv.c](#).

5.5.4.46 sg_httpsrv_is_threaded()

```
bool sg_httpsrv_is_threaded (
    struct sg_httpsrv * srv )
```

Checks if the server was started in threaded model.

Parameters

in	srv	Server handle.
----	-----	----------------

Returns

`true` if the server is in threaded model, `false` otherwise. If **srv** is null, sets the `errno` to `EINVAL`.

5.5.4.47 sg_httpsrv_set_upld_cbs()

```
int sg_httpsrv_set_upld_cbs (
    struct sg_httpsrv * srv,
    sg_httpupld_cb cb,
    void * cls,
    sg_write_cb write_cb,
    sg_free_cb free_cb,
    sg_save_cb save_cb,
    sg_save_as_cb save_as_cb )
```

Sets the server uploading callbacks.

Parameters

in	srv	Server handle.
in	cb	Callback to handle uploaded files and/or fields.
in	cls	User-defined closure.
in	write_cb	Callback to write the stream of the uploaded files.
in	free_cb	Callback to free stream of the uploaded files.
in	save_cb	Callback to save the uploaded files.
in	save_as_cb	Callback to save the uploaded files defining their path.

Return values

0	- Success.
<i>EINVAL</i>	- Invalid argument.

5.5.4.48 sg_httpsrv_set_upld_dir()

```
int sg_httpsrv_set_upld_dir (
    struct sg_httpsrv * srv,
    const char * dir )
```

Sets the directory to save the uploaded files.

Parameters

in	<i>srv</i>	Server handle.
in	<i>dir</i>	Directory as null-terminated string.

Return values

0	- Success.
<i>EINVAL</i>	- Invalid argument.

5.5.4.49 sg_httpsrv_upld_dir()

```
const char* sg_httpsrv_upld_dir (
    struct sg_httpsrv * srv )
```

Gets the directory of the uploaded files.

Parameters

in	<i>srv</i>	Server handle.
----	------------	----------------

Returns

Directory as null-terminated string.

Return values

<i>NULL</i>	If the srv is null and sets the <code>errno</code> to <code>EINVAL</code> .
-------------	--

5.5.4.50 `sg_httpsrv_set_post_buf_size()`

```
int sg_httpsrv_set_post_buf_size (
    struct sg_httpsrv * srv,
    size_t size )
```

Sets a size to the post buffering.

Parameters

in	<i>srv</i>	Server handle.
in	<i>size</i>	Post buffering size.

Return values

0	- Success.
<i>EINVAL</i>	- Invalid argument.

5.5.4.51 `sg_httpsrv_post_buf_size()`

```
size_t sg_httpsrv_post_buf_size (
    struct sg_httpsrv * srv )
```

Gets the size of the post buffering.

Parameters

in	<i>srv</i>	Server handle.
----	------------	----------------

Returns

Post buffering size.

Return values

0	If the srv is null and sets the <code>errno</code> to <code>EINVAL</code> .
---	--

5.5.4.52 `sg_httpsrv_set_payld_limit()`

```
int sg_httpsrv_set_payld_limit (
    struct sg_httpsrv * srv,
    size_t limit )
```

Sets a limit to the total payload.

Parameters

in	<i>srv</i>	Server handle.
in	<i>limit</i>	Payload total limit.

Return values

0	- Success.
<i>EINVAL</i>	- Invalid argument.

5.5.4.53 sg_httpsrv_payld_limit()

```
size_t sg_httpsrv_payld_limit (
    struct sg_httpsrv * srv )
```

Gets the limit of the total payload.

Parameters

in	<i>srv</i>	Server handle.
----	------------	----------------

Returns

Payload total limit.

Return values

0	If the srv is null and sets the <code>errno</code> to <code>EINVAL</code> .
---	--

5.5.4.54 sg_httpsrv_set_uplds_limit()

```
int sg_httpsrv_set_uplds_limit (
    struct sg_httpsrv * srv,
    uint64_t limit )
```

Sets a limit to the total uploads.

Parameters

in	<i>srv</i>	Server handle.
in	<i>limit</i>	Uploads total limit.

Return values

0	- Success.
---	------------

Return values

<i>EINVAL</i>	- Invalid argument.
---------------	---------------------

5.5.4.55 `sg_httpsrv_uplds_limit()`

```
uint64_t sg_httpsrv_uplds_limit (
    struct sg\_httpsrv * srv )
```

Gets the limit of the total uploads.

Parameters

in	<i>srv</i>	Server handle.
----	------------	----------------

Returns

Uploads total limit.

Return values

0	If the <i>srv</i> is null and sets the <code>errno</code> to <code>EINVAL</code> .
---	---

5.5.4.56 `sg_httpsrv_set_thr_pool_size()`

```
int sg_httpsrv_set_thr_pool_size (
    struct sg\_httpsrv * srv,
    unsigned int size )
```

Sets the size for the thread pool.

Parameters

in	<i>srv</i>	Server handle.
in	<i>size</i>	Thread pool size.

Return values

0	- Success.
<i>EINVAL</i>	- Invalid argument.

Examples:

[example_httpsrv_benchmark.c](#).

5.5.4.57 sg_httpsrv_thr_pool_size()

```
unsigned int sg_httpsrv_thr_pool_size (
    struct sg_httpsrv * srv )
```

Gets the size of the thread pool.

Parameters

in	<i>srv</i>	Server handle.
----	------------	----------------

Returns

Thread pool size.

Return values

0	If the srv is null and sets the <code>errno</code> to <code>EINVAL</code> .
---	--

5.5.4.58 sg_httpsrv_set_con_timeout()

```
int sg_httpsrv_set_con_timeout (
    struct sg_httpsrv * srv,
    unsigned int timeout )
```

Sets the inactivity time to a client get time out.

Parameters

in	<i>srv</i>	Server handle.
in	<i>timeout</i>	Timeout (in seconds).

Return values

0	- Success.
<i>EINVAL</i>	- Invalid argument.

5.5.4.59 sg_httpsrv_con_timeout()

```
unsigned int sg_httpsrv_con_timeout (
    struct sg_httpsrv * srv )
```

Gets the inactivity time to a client get time out.

Parameters

in	<i>srv</i>	Server handle.
----	------------	----------------

Returns

Timeout (in seconds).

Return values

0	If the <i>srv</i> is null and sets the <code>errno</code> to <code>EINVAL</code> .
---	---

5.5.4.60 `sg_httpsrv_set_con_limit()`

```
int sg_httpsrv_set_con_limit (  
    struct sg\_httpsrv * srv,  
    unsigned int limit )
```

Sets the limit of concurrent connections.

Parameters

in	<i>srv</i>	Server handle.
in	<i>limit</i>	Concurrent connections limit.

Return values

0	- Success.
<i>EINVAL</i>	- Invalid argument.

Examples:

[example_httpsrv_benchmark.c](#).

5.5.4.61 `sg_httpsrv_con_limit()`

```
unsigned int sg_httpsrv_con_limit (  
    struct sg\_httpsrv * srv )
```

Gets the limit of concurrent connections.

Parameters

in	<i>srv</i>	Server handle.
----	------------	----------------

Returns

Concurrent connections limit.

Return values

0	If the srv is null and sets the <code>errno</code> to <code>EINVAL</code> .
---	--

5.5.4.62 sg_httpread_end()

```
ssize_t sg_httpread_end (
    bool err )
```

Returns a value to end a stream reading processed by [sg_httpres_sendstream\(\)](#).

Parameters

in	<i>err</i>	true to return a value indicating a stream reading error.
----	------------	---

Returns

Value to end a stream reading.

5.6 Path routing

Data Structures

- struct [sg_entrypoint](#)
- struct [sg_entrypoints](#)
- struct [sg_route](#)
- struct [sg_router](#)

Typedefs

- typedef int(* [sg_entrypoints_iter_cb](#)) (void *cls, struct [sg_entrypoint](#) *entrypoint)
- typedef int(* [sg_get_segments_cb](#)) (void *cls, const char *segment)
- typedef int(* [sg_get_vars_cb](#)) (void *cls, const char *name, const char *val)
- typedef void(* [sg_route_cb](#)) (void *cls, struct [sg_route](#) *route)
- typedef int(* [sg_routes_iter_cb](#)) (void *cls, struct [sg_route](#) *route)
- typedef int(* [sg_router_dispatch_cb](#)) (void *cls, const char *path, struct [sg_route](#) *route)
- typedef int(* [sg_router_match_cb](#)) (void *cls, struct [sg_route](#) *route)

Functions

- const char * [sg_entrypoint_name](#) (struct [sg_entrypoint](#) *entrypoint)
- int [sg_entrypoint_set_user_data](#) (struct [sg_entrypoint](#) *entrypoint, void *data)
- void * [sg_entrypoint_user_data](#) (struct [sg_entrypoint](#) *entrypoint)
- struct [sg_entrypoints](#) * [sg_entrypoints_new](#) (void) __attribute__((malloc))
- void [sg_entrypoints_free](#) (struct [sg_entrypoints](#) *entrypoints)
- int [sg_entrypoints_add](#) (struct [sg_entrypoints](#) *entrypoints, const char *path, void *user_data)
- int [sg_entrypoints_rm](#) (struct [sg_entrypoints](#) *entrypoints, const char *path)
- int [sg_entrypoints_iter](#) (struct [sg_entrypoints](#) *entrypoints, [sg_entrypoints_iter_cb](#) cb, void *cls)
- int [sg_entrypoints_clear](#) (struct [sg_entrypoints](#) *entrypoints)
- int [sg_entrypoints_find](#) (struct [sg_entrypoints](#) *entrypoints, struct [sg_entrypoint](#) **entrypoint, const char *path)
- void * [sg_route_handle](#) (struct [sg_route](#) *route)
- void * [sg_route_match](#) (struct [sg_route](#) *route)
- const char * [sg_route_rawpattern](#) (struct [sg_route](#) *route)
- char * [sg_route_pattern](#) (struct [sg_route](#) *route) __attribute__((malloc))
- const char * [sg_route_path](#) (struct [sg_route](#) *route)
- int [sg_route_get_segments](#) (struct [sg_route](#) *route, [sg_get_segments_cb](#) cb, void *cls)
- int [sg_route_get_vars](#) (struct [sg_route](#) *route, [sg_get_vars_cb](#) cb, void *cls)
- void * [sg_route_user_data](#) (struct [sg_route](#) *route)
- int [sg_routes_add2](#) (struct [sg_route](#) **routes, struct [sg_route](#) **route, const char *pattern, char *errmsg, size_t errlen, [sg_route_cb](#) cb, void *cls)
- bool [sg_routes_add](#) (struct [sg_route](#) **routes, const char *pattern, [sg_route_cb](#) cb, void *cls)
- int [sg_routes_rm](#) (struct [sg_route](#) **routes, const char *pattern)
- int [sg_routes_iter](#) (struct [sg_route](#) **routes, [sg_routes_iter_cb](#) cb, void *cls)
- int [sg_routes_next](#) (struct [sg_route](#) **route)
- unsigned int [sg_routes_count](#) (struct [sg_route](#) *routes)
- int [sg_routes_cleanup](#) (struct [sg_route](#) **routes)
- struct [sg_router](#) * [sg_router_new](#) (struct [sg_route](#) *routes) __attribute__((malloc))
- void [sg_router_free](#) (struct [sg_router](#) *router)
- int [sg_router_dispatch2](#) (struct [sg_router](#) *router, const char *path, void *user_data, [sg_router_dispatch_cb](#) dispatch_cb, void *cls, [sg_router_match_cb](#) match_cb)
- int [sg_router_dispatch](#) (struct [sg_router](#) *router, const char *path, void *user_data)

5.6.1 Detailed Description

High-performance path routing.

5.6.2 Typedef Documentation

5.6.2.1 `sg_entrypoints_iter_cb`

```
typedef int(* sg_entrypoints_iter_cb) (void *cls, struct sg\_entrypoint *entrypoint)
```

Callback signature used by [sg_entrypoints_iter\(\)](#) to iterate entry-point items.

Parameters

out	<i>cls</i>	User-defined closure.
out	<i>pair</i>	Current iterated entry-point.

Return values

<i>0</i>	- Success.
<i>E<ERROR></i>	- User-defined error to stop the items iteration.

5.6.2.2 `sg_get_segments_cb`

```
typedef int(* sg_get_segments_cb) (void *cls, const char *segment)
```

Callback signature used by [sg_route_get_segments\(\)](#) to iterate the path segments.

Parameters

out	<i>cls</i>	User-defined closure.
out	<i>segment</i>	Current iterated segment.

Return values

<i>0</i>	- Success.
<i>E<ERROR></i>	- User-defined error to stop the segments iteration.

5.6.2.3 `sg_get_vars_cb`

```
typedef int(* sg_get_vars_cb) (void *cls, const char *name, const char *val)
```

Callback signature used by [sg_route_get_vars\(\)](#) to iterate the path variables.

Parameters

out	<i>cls</i>	User-defined closure.
out	<i>name</i>	Current iterated variable name.
out	<i>val</i>	Current iterated variable value.

Return values

<i>0</i>	- Success.
<i>E<ERROR></i>	- User-defined error to stop the variables iteration.

5.6.2.4 `sg_route_cb`

```
typedef void(* sg_route_cb) (void *cls, struct sg_route *route)
```

Callback signature used to handle the path routing.

Parameters

out	<i>cls</i>	User-defined closure.
out	<i>route</i>	Route handle.

5.6.2.5 `sg_routes_iter_cb`

```
typedef int(* sg_routes_iter_cb) (void *cls, struct sg_route *route)
```

Callback signature used by `sg_routes_iter()` to iterate route items.

Parameters

out	<i>cls</i>	User-defined closure.
out	<i>route</i>	Current iterated route.

Return values

<i>0</i>	- Success.
<i>E<ERROR></i>	- User-defined error to stop the route items iteration.

5.6.2.6 `sg_router_dispatch_cb`

```
typedef int(* sg_router_dispatch_cb) (void *cls, const char *path, struct sg_route *route)
```

Callback signature used by `sg_router_dispatch2` in the route dispatching loop.

Parameters

out	<i>cls</i>	User-defined closure.
out	<i>path</i>	Route path as null-terminated string.
out	<i>route</i>	Route handle.

Return values

<i>0</i>	- Success.
<i>E<ERROR></i>	- User-defined error to stop the route dispatching loop.

5.6.2.7 sg_router_match_cb

```
typedef int(* sg_router_match_cb) (void *cls, struct sg_route *route)
```

Callback signature used by [sg_router_dispatch2](#) when the path matches the pattern before the route dispatching.

Parameters

out	<i>cls</i>	User-defined closure.
out	<i>route</i>	Route handle.

Return values

<i>0</i>	- Success.
<i>E<ERROR></i>	- User-defined error to stop the route dispatching.

5.6.3 Function Documentation**5.6.3.1 sg_entrpoint_name()**

```
const char* sg_entrpoint_name (
    struct sg_entrpoint * entrpoint )
```

Returns the name of the entry-point handle **entrpoint**.

Parameters

in	<i>entrpoint</i>	Entry-point handle.
----	------------------	---------------------

Returns

Entry-point name as null-terminated string.

Return values

<i>NULL</i>	If the entrypoint is null and sets the <code>errno</code> to <code>EINVAL</code> .
-------------	---

5.6.3.2 `sg_entrypoint_set_user_data()`

```
int sg_entrypoint_set_user_data (
    struct sg_entrypoint * entrypoint,
    void * data )
```

Sets user data to the entry-point handle.

Parameters

in	<i>entrypoint</i>	Entry-point handle.
in	<i>data</i>	User data pointer.

Return values

<i>0</i>	- Success.
<i>EINVAL</i>	- Invalid argument.

5.6.3.3 `sg_entrypoint_user_data()`

```
void* sg_entrypoint_user_data (
    struct sg_entrypoint * entrypoint )
```

Gets user data from the entry-point handle.

Parameters

in	<i>entrypoint</i>	Entry-point handle.
----	-------------------	---------------------

Returns

User data pointer.

Return values

<i>NULL</i>	If entrypoint is null and sets the <code>errno</code> to <code>EINVAL</code> .
-------------	---

Examples:

[example_entrypoint.c](#).

5.6.3.4 `sg_entrypoints_new()`

```
struct sg\_entrypoints* sg_entrypoints_new (
    void )
```

Creates a new entry-points handle.

Returns

Entry-points handle.

Warning

It exits the application if called when no memory space is available.

Examples:

[example_entrypoint.c](#).

5.6.3.5 `sg_entrypoints_free()`

```
void sg_entrypoints_free (
    struct sg\_entrypoints * entrypoints )
```

Frees the entry-points handle previously allocated by [sg_entrypoints_new\(\)](#).

Parameters

in	<i>entrypoints</i>	Pointer of the entry-points to be freed.
----	--------------------	--

Examples:

[example_entrypoint.c](#).

5.6.3.6 `sg_entrypoints_add()`

```
int sg_entrypoints_add (
    struct sg\_entrypoints * entrypoints,
    const char * path,
    void * user_data )
```

Adds a new entry-point item to the entry-points **entrypoints**.

Parameters

in	<i>entrypoints</i>	Entry-points handle.
in	<i>path</i>	Entry-point path.
in	<i>user_data</i>	User data pointer.

Return values

<i>0</i>	- Success.
<i>EINVAL</i>	- Invalid argument.
<i>EALREADY</i>	- Entry-point already added.

Warning

It exits the application if called when no memory space is available.

Examples:

[example_entrpoint.c](#).

5.6.3.7 `sg_entrpoints_rm()`

```
int sg_entrpoints_rm (
    struct sg_entrpoints * entrpoints,
    const char * path )
```

Removes an entry-point item from the entry-points **entrpoints**.

Parameters

in	<i>entrpoints</i>	Entry-points handle.
in	<i>path</i>	Entry-point path to be removed.

Return values

<i>0</i>	- Success.
<i>EINVAL</i>	- Invalid argument.
<i>ENOENT</i>	- Entry-point already removed.

Warning

It exits the application if called when no memory space is available.

5.6.3.8 `sg_entrpoints_iter()`

```
int sg_entrpoints_iter (
    struct sg_entrpoints * entrpoints,
    sg_entrpoints_iter_cb cb,
    void * cls )
```

Iterates over entry-point items.

Parameters

in	<i>entrypoints</i>	Entry-points handle.
in	<i>cb</i>	Callback to iterate the entry-point items.
in, out	<i>cls</i>	User-specified value.

Return values

0	- Success.
<i>EINVAL</i>	- Invalid argument.

Returns

Callback result when it is different from 0.

5.6.3.9 sg_entrypoints_clear()

```
int sg_entrypoints_clear (
    struct sg_entrypoints * entrypoints )
```

Cleans all existing entry-point items in the entry-points **entrypoints**.

Parameters

in	<i>entrypoints</i>	Entry-points handle.
----	--------------------	----------------------

Return values

0	- Success.
<i>EINVAL</i>	- Invalid argument.

5.6.3.10 sg_entrypoints_find()

```
int sg_entrypoints_find (
    struct sg_entrypoints * entrypoints,
    struct sg_entrypoint ** entrypoint,
    const char * path )
```

Finds an entry-point item by path.

Parameters

in	<i>entrypoints</i>	Entry-points handle.
in, out	<i>entrypoint</i>	Pointer of the variable to store the found entry-point.
in	<i>path</i>	Entry-point path to be found.

Return values

<i>0</i>	- Success.
<i>EINVAL</i>	- Invalid argument.
<i>ENOENT</i>	- Pair not found.

Warning

It exits the application if called when no memory space is available.

Examples:

[example_entrypoint.c](#).

5.6.3.11 `sg_route_handle()`

```
void* sg_route_handle (
    struct sg_route * route )
```

Returns the PCRE2 handle containing the compiled regex code.

Parameters

in	<i>route</i>	Route handle.
----	--------------	---------------

Returns

PCRE2 handle containing the compiled regex code.

Return values

<i>NULL</i>	If <i>route</i> is null and sets the <code>errno</code> to <code>EINVAL</code> .
-------------	---

5.6.3.12 `sg_route_match()`

```
void* sg_route_match (
    struct sg_route * route )
```

Returns the PCRE2 match data created from the route pattern.

Parameters

in	<i>route</i>	Route handle.
----	--------------	---------------

Returns

PCRE2 match data.

Return values

<i>NULL</i>	If route is null and sets the <code>errno</code> to <code>EINVAL</code> .
-------------	--

5.6.3.13 sg_route_rawpattern()

```
const char* sg_route_rawpattern (
    struct sg_route * route )
```

Returns the raw route pattern. For example, given a pattern `/foo`, the raw pattern is `^/foo$`.

Parameters

in	<i>route</i>	Route handle.
----	--------------	---------------

Returns

Raw pattern as null-terminated string.

Return values

<i>NULL</i>	If route is null and sets the <code>errno</code> to <code>EINVAL</code> .
-------------	--

5.6.3.14 sg_route_pattern()

```
char* sg_route_pattern (
    struct sg_route * route )
```

Returns the route pattern.

Parameters

in	<i>route</i>	Route handle.
----	--------------	---------------

Returns

Pattern as null-terminated string.

Return values

<i>NULL</i>	If route is null and sets the <code>errno</code> to <code>EINVAL</code> .
<i>NULL</i>	If no memory space available and sets the <code>errno</code> to <code>ENOMEM</code> .

Warning

The caller must free the returned value.

5.6.3.15 sg_route_path()

```
const char* sg_route_path (
    struct sg_route * route )
```

Returns the route path.

Parameters

in	<i>route</i>	Route handle.
----	--------------	---------------

Returns

Path component as null-terminated string.

Return values

<i>NULL</i>	If <i>route</i> is null and sets the <i>errno</i> to <i>EINVAL</i> .
-------------	---

Examples:

[example_entrpoint.c](#), [example_router_segments.c](#), [example_router_simple.c](#), and [example_router_vars.c](#).

5.6.3.16 sg_route_get_segments()

```
int sg_route_get_segments (
    struct sg_route * route,
    sg_get_segments_cb cb,
    void * cls )
```

Iterates over path segments.

Parameters

in	<i>route</i>	Route handle.
in	<i>cb</i>	Callback to iterate the path segments.
in, out	<i>cls</i>	User-specified value.

Return values

<i>0</i>	- Success.
<i>EINVAL</i>	- Invalid argument.

Returns

Callback result when it is different from 0.

Examples:

[example_router_segments.c](#).

5.6.3.17 sg_route_get_vars()

```
int sg_route_get_vars (
    struct sg_route * route,
    sg_get_vars_cb cb,
    void * cls )
```

Iterates over path variables.

Parameters

in	<i>route</i>	Route handle.
in	<i>cb</i>	Callback to iterate the path variables.
in, out	<i>cls</i>	User-specified value.

Return values

0	- Success.
<i>EINVAL</i>	- Invalid argument.

Returns

Callback result when it is different from 0.

Examples:

[example_router_srv.c](#), and [example_router_vars.c](#).

5.6.3.18 sg_route_user_data()

```
void* sg_route_user_data (
    struct sg_route * route )
```

Gets user data from the route handle.

Parameters

in	<i>route</i>	Route handle.
----	--------------	---------------

Returns

User data pointer.

Return values

<code>NULL</code>	If <code>route</code> is null and sets the <code>errno</code> to <code>EINVAL</code> .
-------------------	--

Examples:

[example_router_srv.c](#).

5.6.3.19 `sg_routes_add2()`

```
int sg_routes_add2 (
    struct sg_route ** routes,
    struct sg_route ** route,
    const char * pattern,
    char * errmsg,
    size_t errlen,
    sg_route_cb cb,
    void * cls )
```

Adds a route item to the route list `routes`.

Parameters

in, out	<code>routes</code>	Route list pointer to add a new route item.
in, out	<code>route</code>	Pointer of the variable to store the added route reference.
in	<code>pattern</code>	Pattern as null-terminated string. It must be a valid regular expression in PCRE2 syntax.
in, out	<code>errmsg</code>	Pointer of a string to store the error message.
in	<code>errlen</code>	Length of the error message.
in	<code>cb</code>	Callback to handle the path routing.
in	<code>cls</code>	User-defined closure.

Return values

<code>0</code>	- Success.
<code>EINVAL</code>	- Invalid argument.
<code>EALREADY</code>	- Route already added.

Note

The pattern is enclosed between `^` and `$` automatically if it does not start with `(`.
 The escape sequence `\K` is not supported. It causes `EINVAL` if used.
 The pattern is compiled using just-in-time optimization (JIT) when it is supported.

Warning

It exits the application if called when no memory space is available.

5.6.3.20 sg_routes_add()

```
bool sg_routes_add (
    struct sg_route ** routes,
    const char * pattern,
    sg_route_cb cb,
    void * cls )
```

Adds a route item to the route list **routes**. It uses the `stderr` to print the validation errors.

Parameters

in, out	<i>routes</i>	Route list pointer to add a new route item.
in	<i>pattern</i>	Pattern as null-terminated string. It must be a valid regular expression in PCRE2 syntax.
in	<i>cb</i>	Callback to handle the path routing.
in	<i>cls</i>	User-defined closure.

Return values

<i>0</i>	- Success.
<i>EINVAL</i>	- Invalid argument.
<i>EALREADY</i>	- Route already added.

Note

The pattern is enclosed between `^` and `$` automatically if it does not start with `(`.
 The escape sequence `\K` is not supported. It causes `EINVAL` if used.
 The pattern is compiled using just-in-time optimization (JIT) when it is supported.

Warning

It exits the application if called when no memory space is available.

Examples:

[example_entrypoint.c](#), [example_router_segments.c](#), [example_router_simple.c](#), [example_router_srv.c](#), and [example_router_vars.c](#).

5.6.3.21 sg_routes_rm()

```
int sg_routes_rm (
    struct sg_route ** routes,
    const char * pattern )
```

Removes a route item from the route list **routes**.

Parameters

in, out	<i>routes</i>	Route list pointer to add a new route item.
in	<i>pattern</i>	Pattern as null-terminated string of the route to be removed.

Return values

0	- Success.
<i>EINVAL</i>	- Invalid argument.
<i>ENOENT</i>	- Route already removed.

5.6.3.22 `sg_routes_iter()`

```
int sg_routes_iter (
    struct sg_route * routes,
    sg_routes_iter_cb cb,
    void * cls )
```

Iterates over all the routes in the route list **routes**.

Parameters

in	<i>routes</i>	Route list handle.
in	<i>cb</i>	Callback to iterate over route items.
in	<i>cls</i>	User-defined closure.

Return values

0	- Success.
<i>EINVAL</i>	- Invalid argument.
<i>E<ERROR></i>	- User-defined error to abort the list iteration.

5.6.3.23 `sg_routes_next()`

```
int sg_routes_next (
    struct sg_route ** route )
```

Returns the next route int the route list.

Parameters

in, out	<i>route</i>	Pointer to the next route item.
---------	--------------	---------------------------------

Return values

0	- Success.
---	------------

Return values

<i>EINVAL</i>	- Invalid argument.
---------------	---------------------

5.6.3.24 `sg_routes_count()`

```
unsigned int sg_routes_count (
    struct sg_route * routes )
```

Counts the total routes in the route list ***routes***.

Parameters

in	<i>routes</i>	Route list handle.
----	---------------	--------------------

Returns

Total of route items.

Return values

0	If the list is empty or null.
---	-------------------------------

5.6.3.25 `sg_routes_cleanup()`

```
int sg_routes_cleanup (
    struct sg_route ** routes )
```

Cleans the entire route list.

Parameters

in, out	<i>routes</i>	Pointer to the route list.
---------	---------------	----------------------------

Examples:

[example_entrypoint.c](#), [example_router_segments.c](#), [example_router_simple.c](#), [example_router_srv.c](#), and [example_router_vars.c](#).

5.6.3.26 `sg_router_new()`

```
struct sg_router* sg_router_new (
    struct sg_route * routes )
```

Creates a new path router handle. It requires a filled route list ***routes***.

Parameters

in	<i>routes</i>	Route list handle.
----	---------------	--------------------

Returns

New router handle.

Return values

<i>NULL</i>	If the routes is null and sets the <code>errno</code> to <code>EINVAL</code> .
-------------	---

Examples:

[example_entrpoint.c](#), [example_router_segments.c](#), [example_router_simple.c](#), [example_router_srv.c](#), and [example_router_vars.c](#).

5.6.3.27 `sg_router_free()`

```
void sg_router_free (
    struct sg_router * router )
```

Frees the router handle previously allocated by `sg_router_new()`.

Parameters

in	<i>router</i>	Router handle.
----	---------------	----------------

Examples:

[example_entrpoint.c](#), [example_router_segments.c](#), [example_router_simple.c](#), [example_router_srv.c](#), and [example_router_vars.c](#).

5.6.3.28 `sg_router_dispatch2()`

```
int sg_router_dispatch2 (
    struct sg_router * router,
    const char * path,
    void * user_data,
    sg_router_dispatch_cb dispatch_cb,
    void * cls,
    sg_router_match_cb match_cb )
```

Dispatches a route that its pattern matches the path passed in **path**.

Parameters

in	<i>router</i>	Router handle.
in	<i>path</i>	Path to dispatch a route.
in	<i>user_data</i>	User data pointer to be hold by the route.
in	<i>dispatch_cb</i>	Callback triggered for each route item in the route dispatching loop.
in	<i>cls</i>	User-defined closure passed to the dispatch_cb and match_cb callbacks.
in	<i>match_cb</i>	Callback triggered when the path matches the route pattern.

Return values

0	- Success.
<i>EINVAL</i>	- Invalid argument.
<i>ENOENT</i>	- Route not found or path not matched.
<i>E<ERROR></i>	- User-defined error in dispatch_cb or match_cb .

Note

The route callback [sg_route_cb](#) is triggered when the path matches the route pattern.
The match logic uses just-in-time optimization (JIT) when it is supported.

5.6.3.29 sg_router_dispatch()

```
int sg_router_dispatch (
    struct sg_router * router,
    const char * path,
    void * user_data )
```

Dispatches a route that its pattern matches the path passed in **path**.

Parameters

in	<i>router</i>	Router handle.
in	<i>path</i>	Path to dispatch a route.
in	<i>user_data</i>	User data pointer to be hold by the route.

Return values

0	- Success.
<i>EINVAL</i>	- Invalid argument.
<i>ENOENT</i>	- Route not found or path not matched.
<i>E<ERROR></i>	- User-defined error in dispatch_cb or match_cb .

Note

The route callback [sg_route_cb](#) is triggered when the path matches the route pattern.
The match logic uses just-in-time optimization (JIT) when it is supported.

Examples:

[example_entrypoint.c](#), [example_router_segments.c](#), [example_router_simple.c](#), [example_router_srv.c](#), and [example_router_vars.c](#).

6 Data Structure Documentation

6.1 `sg_entrypoint` Struct Reference

```
#include <sagui.h>
```

6.1.1 Detailed Description

Handle for the entry-point handling. It defines an entry-point to a path or resource. For example, given a path `/api1/customer`, the part considered as entry-point is `/api1`.

Examples:

[example_entrypoint.c](#).

The documentation for this struct was generated from the following file:

- [sagui.h](#)

6.2 `sg_entrypoints` Struct Reference

```
#include <sagui.h>
```

6.2.1 Detailed Description

Handle for the entry-point list. It is used to create a list of entry-point [sg_entrypoint](#).

Examples:

[example_entrypoint.c](#).

The documentation for this struct was generated from the following file:

- [sagui.h](#)

6.3 `sg_httpauth` Struct Reference

```
#include <sagui.h>
```

6.3.1 Detailed Description

Handle for the HTTP basic authentication.

Examples:

[example_httpauth.c](#).

The documentation for this struct was generated from the following file:

- [sagui.h](#)

6.4 `sg_httpreq` Struct Reference

```
#include <sagui.h>
```

6.4.1 Detailed Description

Handle for the request handling. It contains headers, cookies, query-string, fields, payloads, uploads and other data sent by the client.

Examples:

[example_httpauth.c](#), [example_httpcookie.c](#), [example_httpsrv.c](#), [example_httpsrv_benchmark.c](#), [example_httpsrv_tls.c](#), [example_httpsrv_tls_cert_auth.c](#), [example_httpuplds.c](#), and [example_router_srv.c](#).

The documentation for this struct was generated from the following file:

- [sagui.h](#)

6.5 `sg_httpres` Struct Reference

```
#include <sagui.h>
```

6.5.1 Detailed Description

Handle for the response handling. It dispatches headers, contents, binaries, files and other data to the client.

Examples:

[example_httpauth.c](#), [example_httpcookie.c](#), [example_httpsrv.c](#), [example_httpsrv_benchmark.c](#), [example_httpsrv_tls.c](#), [example_httpsrv_tls_cert_auth.c](#), [example_httpuplds.c](#), and [example_router_srv.c](#).

The documentation for this struct was generated from the following file:

- [sagui.h](#)

6.6 `sg_httpsrv` Struct Reference

```
#include <sagui.h>
```

6.6.1 Detailed Description

Handle for the fast event-driven HTTP server.

Examples:

[example_httpauth.c](#), [example_httpcookie.c](#), [example_httpsrv.c](#), [example_httpsrv_benchmark.c](#), [example_httpsrv_tls.c](#), [example_httpsrv_tls_cert_auth.c](#), [example_httpuplds.c](#), and [example_router_srv.c](#).

The documentation for this struct was generated from the following file:

- [sagui.h](#)

6.7 `sg_httpupld` Struct Reference

```
#include <sagui.h>
```

6.7.1 Detailed Description

Handle for the upload handling. It is used to represent a single upload or a list of uploads.

Examples:

[example_httpuplds.c](#).

The documentation for this struct was generated from the following file:

- [sagui.h](#)

6.8 `sg_route` Struct Reference

```
#include <sagui.h>
```

6.8.1 Detailed Description

Handle for the route item. It holds a user data to be dispatched when a path matches the user defined pattern (route pattern).

Examples:

[example_entrpoint.c](#), [example_router_segments.c](#), [example_router_simple.c](#), [example_router_srv.c](#), and [example_router_vars.c](#).

The documentation for this struct was generated from the following file:

- [sagui.h](#)

6.9 `sg_router` Struct Reference

```
#include <sagui.h>
```

6.9.1 Detailed Description

Handle for the path router. It holds the reference of a route list to be dispatched.

Examples:

[example_entrpoint.c](#), [example_router_segments.c](#), [example_router_simple.c](#), [example_router_srv.c](#), and [example_router_vars.c](#).

The documentation for this struct was generated from the following file:

- [sagui.h](#)

6.10 sg_str Struct Reference

```
#include <sagui.h>
```

6.10.1 Detailed Description

Handle for the string structure used to represent a HTML body, POST payload and more.

Examples:

[example_httpuplds.c](#), [example_router_srv.c](#), and [example_str.c](#).

The documentation for this struct was generated from the following file:

- [sagui.h](#)

6.11 sg_strmap Struct Reference

```
#include <sagui.h>
```

6.11.1 Detailed Description

Handle for hash table that maps name-value pairs. It is useful to represent posting fields, query-string parameter, client cookies and more.

Examples:

[example_httpcookie.c](#), [example_httpuplds.c](#), and [example_strmap.c](#).

The documentation for this struct was generated from the following file:

- [sagui.h](#)

7 File Documentation

7.1 `example_entrypoint.h` File Reference

7.2 `example_httpauth.h` File Reference

7.3 `example_httpcookie.h` File Reference

7.4 `example_httpsrv.h` File Reference

7.5 `example_httpsrv_benchmark.h` File Reference

7.6 `example_httpsrv_tls.h` File Reference

7.7 `example_httpsrv_tls_cert_auth.h` File Reference

7.8 `example_httpuplds.h` File Reference

7.9 `example_router_segments.h` File Reference

7.10 `example_router_simple.h` File Reference

7.11 `example_router_srv.h` File Reference

7.12 `example_router_vars.h` File Reference

7.13 `example_str.h` File Reference

7.14 `example_strmap.h` File Reference

7.15 `sagui.h` File Reference

```
#include <stdio.h>
#include <stddef.h>
#include <stdbool.h>
#include <stdint.h>
#include <stdarg.h>
#include <string.h>
#include <time.h>
```

Macros

- `#define SG_ERR_SIZE 256`
- `#define sg_httpres_send(res, val, content_type, status) sg_httpres_sendbinary((res), (void *) (val), ((val != NULL) ? strlen((val)) : 0), (content_type), (status))`

Typedefs

- typedef void(* [sg_err_cb](#)) (void *cls, const char *err)
- typedef size_t(* [sg_write_cb](#)) (void *handle, uint64_t offset, const char *buf, size_t size)
- typedef ssize_t(* [sg_read_cb](#)) (void *handle, uint64_t offset, char *buf, size_t size)
- typedef void(* [sg_free_cb](#)) (void *handle)
- typedef int(* [sg_save_cb](#)) (void *handle, bool overwritten)
- typedef int(* [sg_save_as_cb](#)) (void *handle, const char *path, bool overwritten)
- typedef int(* [sg_strmap_iter_cb](#)) (void *cls, struct [sg_strmap](#) *pair)
- typedef int(* [sg_strmap_sort_cb](#)) (void *cls, struct [sg_strmap](#) *pair_a, struct [sg_strmap](#) *pair_b)
- typedef bool(* [sg_httpauth_cb](#)) (void *cls, struct [sg_httpauth](#) *auth, struct [sg_httpreq](#) *req, struct [sg_httpres](#) *res)
- typedef int(* [sg_httpupld_cb](#)) (void *cls, void **handle, const char *dir, const char *field, const char *name, const char *mime, const char *encoding)
- typedef int(* [sg_httpuplds_iter_cb](#)) (void *cls, struct [sg_httpupld](#) *upld)
- typedef void(* [sg_httpreq_cb](#)) (void *cls, struct [sg_httpreq](#) *req, struct [sg_httpres](#) *res)
- typedef int(* [sg_entrpoints_iter_cb](#)) (void *cls, struct [sg_entrpoint](#) *entrpoint)
- typedef int(* [sg_get_segments_cb](#)) (void *cls, const char *segment)
- typedef int(* [sg_get_vars_cb](#)) (void *cls, const char *name, const char *val)
- typedef void(* [sg_route_cb](#)) (void *cls, struct [sg_route](#) *route)
- typedef int(* [sg_routes_iter_cb](#)) (void *cls, struct [sg_route](#) *route)
- typedef int(* [sg_router_dispatch_cb](#)) (void *cls, const char *path, struct [sg_route](#) *route)
- typedef int(* [sg_router_match_cb](#)) (void *cls, struct [sg_route](#) *route)

Functions

- unsigned int [sg_version](#) (void)
- const char * [sg_version_str](#) (void)
- void * [sg_alloc](#) (size_t size) __attribute__((malloc))
- void * [sg_realloc](#) (void *ptr, size_t size) __attribute__((malloc))
- void [sg_free](#) (void *ptr)
- char * [sg_strerror](#) (int errnum, char *errmsg, size_t errlen)
- bool [sg_is_post](#) (const char *method)
- char * [sg_extract_entrpoint](#) (const char *path)
- char * [sg_tmpdir](#) (void)
- struct [sg_str](#) * [sg_str_new](#) (void) __attribute__((malloc))
- void [sg_str_free](#) (struct [sg_str](#) *str)
- int [sg_str_write](#) (struct [sg_str](#) *str, const char *val, size_t len)
- int [sg_str_printf_va](#) (struct [sg_str](#) *str, const char *fmt, va_list ap)
- int [sg_str_printf](#) (struct [sg_str](#) *str, const char *fmt, ...) __attribute__((format(printf)))
- int const char * [sg_str_content](#) (struct [sg_str](#) *str)
- size_t [sg_str_length](#) (struct [sg_str](#) *str)
- int [sg_str_clear](#) (struct [sg_str](#) *str)
- const char * [sg_strmap_name](#) (struct [sg_strmap](#) *pair)
- const char * [sg_strmap_val](#) (struct [sg_strmap](#) *pair)
- int [sg_strmap_add](#) (struct [sg_strmap](#) **map, const char *name, const char *val)
- int [sg_strmap_set](#) (struct [sg_strmap](#) **map, const char *name, const char *val)
- int [sg_strmap_find](#) (struct [sg_strmap](#) *map, const char *name, struct [sg_strmap](#) **pair)
- const char * [sg_strmap_get](#) (struct [sg_strmap](#) *map, const char *name)
- int [sg_strmap_rm](#) (struct [sg_strmap](#) **map, const char *name)
- int [sg_strmap_iter](#) (struct [sg_strmap](#) *map, [sg_strmap_iter_cb](#) cb, void *cls)
- int [sg_strmap_sort](#) (struct [sg_strmap](#) **map, [sg_strmap_sort_cb](#) cb, void *cls)
- unsigned int [sg_strmap_count](#) (struct [sg_strmap](#) *map)
- int [sg_strmap_next](#) (struct [sg_strmap](#) **next)

- void [sg_strmap_cleanup](#) (struct [sg_strmap](#) **map)
- int [sg_httpauth_set_realm](#) (struct [sg_httpauth](#) *auth, const char *realm)
- const char * [sg_httpauth_realm](#) (struct [sg_httpauth](#) *auth)
- int [sg_httpauth_deny](#) (struct [sg_httpauth](#) *auth, const char *justification, const char *content_type)
- int [sg_httpauth_cancel](#) (struct [sg_httpauth](#) *auth)
- const char * [sg_httpauth_usr](#) (struct [sg_httpauth](#) *auth)
- const char * [sg_httpauth_pwd](#) (struct [sg_httpauth](#) *auth)
- int [sg_httpuplds_iter](#) (struct [sg_httpupld](#) *uplds, [sg_httpuplds_iter_cb](#) cb, void *cls)
- int [sg_httpuplds_next](#) (struct [sg_httpupld](#) **upld)
- unsigned int [sg_httpuplds_count](#) (struct [sg_httpupld](#) *uplds)
- void * [sg_httpupld_handle](#) (struct [sg_httpupld](#) *upld)
- const char * [sg_httpupld_dir](#) (struct [sg_httpupld](#) *upld)
- const char * [sg_httpupld_field](#) (struct [sg_httpupld](#) *upld)
- const char * [sg_httpupld_name](#) (struct [sg_httpupld](#) *upld)
- const char * [sg_httpupld_mime](#) (struct [sg_httpupld](#) *upld)
- const char * [sg_httpupld_encoding](#) (struct [sg_httpupld](#) *upld)
- uint64_t [sg_httpupld_size](#) (struct [sg_httpupld](#) *upld)
- int [sg_httpupld_save](#) (struct [sg_httpupld](#) *upld, bool overwritten)
- int [sg_httpupld_save_as](#) (struct [sg_httpupld](#) *upld, const char *path, bool overwritten)
- struct [sg_strmap](#) ** [sg_httpreq_headers](#) (struct [sg_httpreq](#) *req)
- struct [sg_strmap](#) ** [sg_httpreq_cookies](#) (struct [sg_httpreq](#) *req)
- struct [sg_strmap](#) ** [sg_httpreq_params](#) (struct [sg_httpreq](#) *req)
- struct [sg_strmap](#) ** [sg_httpreq_fields](#) (struct [sg_httpreq](#) *req)
- const char * [sg_httpreq_version](#) (struct [sg_httpreq](#) *req)
- const char * [sg_httpreq_method](#) (struct [sg_httpreq](#) *req)
- const char * [sg_httpreq_path](#) (struct [sg_httpreq](#) *req)
- struct [sg_str](#) * [sg_httpreq_payload](#) (struct [sg_httpreq](#) *req)
- bool [sg_httpreq_is_uploading](#) (struct [sg_httpreq](#) *req)
- struct [sg_httpupld](#) * [sg_httpreq_uploads](#) (struct [sg_httpreq](#) *req)
- void * [sg_httpreq_tls_session](#) (struct [sg_httpreq](#) *req)
- int [sg_httpreq_set_user_data](#) (struct [sg_httpreq](#) *req, void *data)
- void * [sg_httpreq_user_data](#) (struct [sg_httpreq](#) *req)
- struct [sg_strmap](#) ** [sg_httpres_headers](#) (struct [sg_httpres](#) *res)
- int [sg_httpres_set_cookie](#) (struct [sg_httpres](#) *res, const char *name, const char *val)
- int [sg_httpres_sendbinary](#) (struct [sg_httpres](#) *res, void *buf, size_t size, const char *content_type, unsigned int status)
- int [sg_httpres_sendfile](#) (struct [sg_httpres](#) *res, size_t block_size, uint64_t max_size, const char *filename, bool rendered, unsigned int status)
- int [sg_httpres_sendstream](#) (struct [sg_httpres](#) *res, uint64_t size, size_t block_size, [sg_read_cb](#) read_cb, void *handle, [sg_free_cb](#) free_cb, unsigned int status)
- int [sg_httpres_clear](#) (struct [sg_httpres](#) *res)
- struct [sg_httpsrv](#) * [sg_httpsrv_new2](#) ([sg_httpauth_cb](#) auth_cb, void *auth_cls, [sg_httpreq_cb](#) req_cb, void *req_cls, [sg_err_cb](#) err_cb, void *err_cls) __attribute__((malloc))
- struct [sg_httpsrv](#) * [sg_httpsrv_new](#) ([sg_httpreq_cb](#) cb, void *cls) __attribute__((malloc))
- void [sg_httpsrv_free](#) (struct [sg_httpsrv](#) *srv)
- bool [sg_httpsrv_tls_listen2](#) (struct [sg_httpsrv](#) *srv, const char *key, const char *pwd, const char *cert, const char *trust, const char *dhparams, uint16_t port, bool threaded)
- bool [sg_httpsrv_tls_listen](#) (struct [sg_httpsrv](#) *srv, const char *key, const char *cert, uint16_t port, bool threaded)
- bool [sg_httpsrv_listen](#) (struct [sg_httpsrv](#) *srv, uint16_t port, bool threaded)
- int [sg_httpsrv_shutdown](#) (struct [sg_httpsrv](#) *srv)
- uint16_t [sg_httpsrv_port](#) (struct [sg_httpsrv](#) *srv)
- bool [sg_httpsrv_is_threaded](#) (struct [sg_httpsrv](#) *srv)
- int [sg_httpsrv_set_upld_cbs](#) (struct [sg_httpsrv](#) *srv, [sg_httpupld_cb](#) cb, void *cls, [sg_write_cb](#) write_cb, [sg_free_cb](#) free_cb, [sg_save_cb](#) save_cb, [sg_save_as_cb](#) save_as_cb)

- int [sg_httpsrv_set_upld_dir](#) (struct [sg_httpsrv](#) *srv, const char *dir)
- const char * [sg_httpsrv_upld_dir](#) (struct [sg_httpsrv](#) *srv)
- int [sg_httpsrv_set_post_buf_size](#) (struct [sg_httpsrv](#) *srv, size_t size)
- size_t [sg_httpsrv_post_buf_size](#) (struct [sg_httpsrv](#) *srv)
- int [sg_httpsrv_set_payld_limit](#) (struct [sg_httpsrv](#) *srv, size_t limit)
- size_t [sg_httpsrv_payld_limit](#) (struct [sg_httpsrv](#) *srv)
- int [sg_httpsrv_set_uplds_limit](#) (struct [sg_httpsrv](#) *srv, uint64_t limit)
- uint64_t [sg_httpsrv_uplds_limit](#) (struct [sg_httpsrv](#) *srv)
- int [sg_httpsrv_set_thr_pool_size](#) (struct [sg_httpsrv](#) *srv, unsigned int size)
- unsigned int [sg_httpsrv_thr_pool_size](#) (struct [sg_httpsrv](#) *srv)
- int [sg_httpsrv_set_con_timeout](#) (struct [sg_httpsrv](#) *srv, unsigned int timeout)
- unsigned int [sg_httpsrv_con_timeout](#) (struct [sg_httpsrv](#) *srv)
- int [sg_httpsrv_set_con_limit](#) (struct [sg_httpsrv](#) *srv, unsigned int limit)
- unsigned int [sg_httpsrv_con_limit](#) (struct [sg_httpsrv](#) *srv)
- ssize_t [sg_httpread_end](#) (bool err)
- const char * [sg_entrpoint_name](#) (struct [sg_entrpoint](#) *entrpoint)
- int [sg_entrpoint_set_user_data](#) (struct [sg_entrpoint](#) *entrpoint, void *data)
- void * [sg_entrpoint_user_data](#) (struct [sg_entrpoint](#) *entrpoint)
- struct [sg_entrpoints](#) * [sg_entrpoints_new](#) (void) __attribute__((malloc))
- void [sg_entrpoints_free](#) (struct [sg_entrpoints](#) *entrpoints)
- int [sg_entrpoints_add](#) (struct [sg_entrpoints](#) *entrpoints, const char *path, void *user_data)
- int [sg_entrpoints_rm](#) (struct [sg_entrpoints](#) *entrpoints, const char *path)
- int [sg_entrpoints_iter](#) (struct [sg_entrpoints](#) *entrpoints, [sg_entrpoints_iter_cb](#) cb, void *cls)
- int [sg_entrpoints_clear](#) (struct [sg_entrpoints](#) *entrpoints)
- int [sg_entrpoints_find](#) (struct [sg_entrpoints](#) *entrpoints, struct [sg_entrpoint](#) **entrpoint, const char *path)
- void * [sg_route_handle](#) (struct [sg_route](#) *route)
- void * [sg_route_match](#) (struct [sg_route](#) *route)
- const char * [sg_route_rawpattern](#) (struct [sg_route](#) *route)
- char * [sg_route_pattern](#) (struct [sg_route](#) *route) __attribute__((malloc))
- const char * [sg_route_path](#) (struct [sg_route](#) *route)
- int [sg_route_get_segments](#) (struct [sg_route](#) *route, [sg_get_segments_cb](#) cb, void *cls)
- int [sg_route_get_vars](#) (struct [sg_route](#) *route, [sg_get_vars_cb](#) cb, void *cls)
- void * [sg_route_user_data](#) (struct [sg_route](#) *route)
- int [sg_routes_add2](#) (struct [sg_route](#) **routes, struct [sg_route](#) **route, const char *pattern, char *errmsg, size_t errlen, [sg_route_cb](#) cb, void *cls)
- bool [sg_routes_add](#) (struct [sg_route](#) **routes, const char *pattern, [sg_route_cb](#) cb, void *cls)
- int [sg_routes_rm](#) (struct [sg_route](#) **routes, const char *pattern)
- int [sg_routes_iter](#) (struct [sg_route](#) *routes, [sg_routes_iter_cb](#) cb, void *cls)
- int [sg_routes_next](#) (struct [sg_route](#) **route)
- unsigned int [sg_routes_count](#) (struct [sg_route](#) *routes)
- int [sg_routes_cleanup](#) (struct [sg_route](#) **routes)
- struct [sg_router](#) * [sg_router_new](#) (struct [sg_route](#) *routes) __attribute__((malloc))
- void [sg_router_free](#) (struct [sg_router](#) *router)
- int [sg_router_dispatch2](#) (struct [sg_router](#) *router, const char *path, void *user_data, [sg_router_dispatch_cb](#) dispatch_cb, void *cls, [sg_router_match_cb](#) match_cb)
- int [sg_router_dispatch](#) (struct [sg_router](#) *router, const char *path, void *user_data)

7.15.1 Macro Definition Documentation


```
sg_entrypoints_find(entrypoints, &entrypoint, "/r2/foo");
sg_router_dispatch(sg_entrypoint_user_data(entrypoint), "/foo"
, NULL);
sg_entrypoints_find(entrypoints, &entrypoint, "/r2/bar");
sg_router_dispatch(sg_entrypoint_user_data(entrypoint), "/bar"
, NULL);

sg_routes_cleanup(&rts1);
sg_routes_cleanup(&rts2);
sg_router_free(r1);
sg_router_free(r2);
sg_entrypoints_free(entrypoints);
return EXIT_SUCCESS;
}
```

8.2 example_httpauth.c

Simple example showing the Basic authentication feature.

[illegible]

8.5 example_httpsrv_benchmark.c

Simple "hello world" HTTP server to be used by benchmark tests.

```

/*
 *      _
 *  _/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_
 *  \_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_
 *  |_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_|
 *  |_/_/_/_/_/_/_/_/_/_/_/_/_/_/__|
 *
 *  -- an ideal C library to develop cross-platform HTTP servers.
 *
 *  Copyright (c) 2016-2018 Silvio Clecio <silvioprog@gmail.com>
 *
 *  This file is part of Sagui library.
 *
 *  Sagui library is free software: you can redistribute it and/or modify
 *  it under the terms of the GNU Lesser General Public License as published by
 *  the Free Software Foundation, either version 3 of the License, or
 *  (at your option) any later version.
 *
 *  Sagui library is distributed in the hope that it will be useful,
 *  but WITHOUT ANY WARRANTY; without even the implied warranty of
 *  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 *  GNU Lesser General Public License for more details.
 *
 *  You should have received a copy of the GNU Lesser General Public License
 *  along with Sagui library. If not, see <http://www.gnu.org/licenses/>.
 */

#include <stdlib.h>

#ifdef _WIN32
#include <windows.h>
#else
#include <unistd.h>
#endif
#include <sagui.h>

/* NOTE: Error checking has been omitted to make it clear. */

static unsigned int get_cpu_count(void) {
#ifdef _WIN32
#ifdef _SC_NPROCESSORS_ONLN
SYSTEM_INFO info;
GetSystemInfo(&info);
#define sysconf(void) info.dwNumberOfProcessors
#define _SC_NPROCESSORS_ONLN
#endif
#endif
#ifdef _SC_NPROCESSORS_ONLN
return (unsigned int) sysconf(_SC_NPROCESSORS_ONLN);
#else
return 0;
#endif
}

static void req_cb(__SG_UNUSED void *cls, __SG_UNUSED struct sg_httpreq *req, struct
sg_httpsrv *res) {
sg_httpsrv_send(res, "<html><head><title>Hello world</title></head><body>Hello
world</body></html>",
"text/html", 200);
}

int main(void) {
const unsigned int cpu_count = get_cpu_count();
const unsigned int con_limit = 1000; /* Change to 10000 for C10K problem. */
struct sg_httpsrv *srv = sg_httpsrv_new(req_cb, NULL);
sg_httpsrv_set_thr_pool_size(srv, cpu_count);
sg_httpsrv_set_con_limit(srv, con_limit);
if (!sg_httpsrv_listen(srv, 0 /* 0 = port chosen randomly */, false)) {
sg_httpsrv_free(srv);
return EXIT_FAILURE;
}
fprintf(stdout, "Number of processors: %d\n", cpu_count);
fprintf(stdout, "Connections limit: %d\n", con_limit);
fprintf(stdout, "Server running at http://localhost:%d\n", sg_httpsrv_port(srv));
fflush(stdout);
getchar();
sg_httpsrv_free(srv);
return EXIT_SUCCESS;
}

```

8.6 example_httpsrv_tls.c

Simple "hello world" HTTPS server example.

[illegible]

```

"6dXk2CWh37A7ugf/2vyqLzQK+I17GjtcM9S+T7NZNo1Eu+7xYIWF04QCj4/+1/vd\n"
"ezxzikcSgeGG6IkDnyX/Qe2xr40UugPlcLqK2vHNaJnVbGcJD1I4+mKeeCZsDGVt\n"
"QzCET0CpvlpuvUZ+5kyM3hEeLYLOUZ4MDjLT2EU7UBjOV204hC9sdU9fhv8dUxvj\n"
"521LVlysz/08cvyAi+AOpPqPK2dWS6z3HiqAk5HyjvCaMwKBwQDPpRimhEhmAZ0h\n"
"Wm9qt0QcEahfFDYked/FeJgzd3dn8CgFiiObL1j7wYVIV51GmSzeRAdSWwLRQWB\n"
"pmnlnNyxomtweyHgZlYpU5S7tiJlcf196SvNqnwwllr0ZqrFoh8k3UwgKytWvfjV\n"
"orhGklgAl1P2EEiAxS06XYLnHmKn9mq+cLrKxQHAXqb7u+kRgnCXZUVuAW1CdiyI\n"
"cCGRr3RznEH6FkN0hzdtNwvHduj/AB8nAlJVq9X6PWYRhuFGfKgcAOfpQgd2Rj\n"
"tnRIGiGkAKR4Pyn69xdI7xrgDqdpA07rjaINBsXnGZ+xp8whfVh8JZU9VQoVzm2f\n"
"uUwHKMGogYAonaMjeFoZl7QGS5/LVBPz9VTVD8VVY5EQr7Mh3kLUSC2h1RRdfdyE9\n"
"RVpU2POvbFwetMVh2Q18/4i4vd02khzbn9u0JeUktVGUbVAP16IcOP1Vy9h2BseG\n"
"8WwEjhs93VRNy/PSxmAeVYymaDSqR5eBL+/eExk+ryr93InlaQmj5Is=\n"
"-----END RSA PRIVATE KEY-----";

const char certificate[] =
"-----BEGIN CERTIFICATE-----\n"
"MIIEODCCAqCgAwIBAgIMW23FvhiQ/Xip31BxMA0GCSqGSIb3DQEBCwUAMDCxGDAW\n"
"BgnVBAMTD3Rlc3QuZ251dGxzLm9yZzEbmBkGA1UEChMSR251VExTIHRlc3Qgc2Vy\n"
"dmVyMB4XDTE4MDgxMDE3MDUwMl0XDTI4MDgwNzE3MDUwMl0wNzEYMBYGA1UEAxMP\n"
"dGVzdC5nbmV0bHMub3JnMRswGQYDVQQKEzJHbnVUTFMgdGVzdCBzZXJ2ZXIwggGi\n"
"MA0GCSqGSIb3DQEBAQUAA4IBjwAwggGKAoIBgQDWrNobZBvCcHNTfGQ0LuhXsGg0\n"
"zVchXg8fTfQUkvvsPH4p8v+p9TBjo2TPL+IO2fUFitSZ3wy6zfihuwLRp3+whqtq\n"
"KUr/+YBhXktUe4uA4nhzUWgyoy9p8QHrsssXF4i3ArVfvxpP1NIRs5mqm5VeYQ87\n"
"RW97nedAdpGvvMftvSPy0GrEJzTvJb1NGWg5y2GiVhKnWY+JoAnl39ghyZIR4Po0\n"
"GkxnvWC4cg1vrmlJkTnf42UkdJWNxWCrUCnmCDr5bZJjpNJKNtuccA2ahUym7r3S\n"
"M/olcMg5jkgiecxKT6rFviFADxr+B4sqigoi3Dzm06Xbtv5qnfGPANlKuLfiXmJ5\n"
"FqTK6cGKmcT1tLPR2aXcqI8z7qFsCuyQ+58PeCMeMGFCqWg8DSKacaufj+vCZpMW\n"
"mUu/eQJ/JMSVBwxHuKQXm502fjJnMPMDuB5Y8E5nZd42Qrl/5Qs5+m/FmxhFoJIM\n"
"s+1jb/DCMu9KWPfPaSaY6eJ+vg0V2n94V89A8KgcCAwEAAaNEMEIwDAYDVDR0TAQH\n"
"BA1wADATBgNVHSUEDDAKBggrBgEFBQcDATAdBgNVHQ4EFgQUxH1HUKpvYFEhrPeJ\n"
"sYOI7HGDblIwDQYJKoZIhvcNAQELBQADggGBABDTlhiKuuh51Rx+mpt5vjJ7zXRJ\n"
"lRoHY92AmZY49hfdpUp6mMLvbRdD6REv4pelIORGsggPk4MPCQsaFbGpZS1CokLz\n"
"Sex4LZuHuYwtv18fUFl1JIZZSITnArNB291XAem5T20D04bCCYLJJB3VtcPilbkf\n"
"ipT/hlCyhbWX14ZtkzpwMAwLgod6uZvJqJXTPjwdWA7Anp4yfh2QxBYC5/us4xP\n"
"wHa0euWOBZ+Q9ZNZ/fFdLESLSbBob9736hBg1NSgBFCMNezqs18/EGIJcS7w96PN\n"
"YJcVsVhcQUMMT4dnaSM/Ri4CPv7j8/z1l6uq4kHpxZLhuxerSeuKBn6j10wHQFdl\n"
"7bpHrRLBuRyDhPwrzdmY2dyJ5DkO39auiSAYJa8IddfNnCa7howSjp/ZvZN9Sf\n"
"gilk1ZeSpe+iiJWQaxjIKAr/g8Rn+ALfeMAitm6DjCcTUKxKdTxVqTwdFZRNxrNH\n"
"lqt+H07raUsv/p50oVS6/Euv8fBm3EKPwx64w==\n"
"-----END CERTIFICATE-----";

static void req_cb(__SG_UNUSED void *cls, __SG_UNUSED struct sg_httpreq *req, struct
sg_httpsres *res) {
sg_httpsres_send(res,
" <html><head><title>Hello world</title></head><body>Hello <font color=\"green\">
>HTTPS</font></body></html>",
"text/html; charset=utf-8", 200);
}

int main(void) {
struct sg_httpsrv *srv = sg_httpsrv_new(req_cb, NULL);
if (!sg_httpsrv_tls_listen(srv, private_key, certificate, 0 /* 0 = port chosen
randomly */, false)) {
sg_httpsrv_free(srv);
return EXIT_FAILURE;
}
fprintf(stdout, "Server running at https://localhost:%d\n", sg_httpsrv_port(srv));
fflush(stdout);
getchar();
sg_httpsrv_free(srv);
return EXIT_SUCCESS;
}

```

8.7 example_httpsrv_tls_cert_auth.c

Simple client-side certificate authentication using GnuTLS.

```

/*
 *
 *  / _ _ \ / _ _ \ / _ _ \ / _ _ \
 *  \ _ _ \ / _ _ \ / _ _ \ / _ _ \
 *  / _ _ \ / _ _ \ / _ _ \ / _ _ \
 *  / _ _ \ / _ _ \ / _ _ \ / _ _ \
 *
 *  -- an ideal C library to develop cross-platform HTTP servers.
 *
 *  Copyright (c) 2016-2018 Silvio Clecio <silvioprog@gmail.com>
 *
 *  This file is part of Sagui library.
 *
 *  Sagui library is free software: you can redistribute it and/or modify

```

```

* it under the terms of the GNU Lesser General Public License as published by
* the Free Software Foundation, either version 3 of the License, or
* (at your option) any later version.
*
* Saguï library is distributed in the hope that it will be useful,
* but WITHOUT ANY WARRANTY; without even the implied warranty of
* MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
* GNU Lesser General Public License for more details.
*
* You should have received a copy of the GNU Lesser General Public License
* along with Saguï library. If not, see <http://www.gnu.org/licenses/>.
*/

#include <stdlib.h>
#include <stdio.h>
#include <errno.h>
#include <gnutls/gnutls.h>
#include <gnutls/x509.h>
#include <saguï.h>

/* NOTE: Error checking has been omitted to make it clear. */

/*
 * Simple example using TLS client authentication.
 */
/*
 * Client example using cURL:
 */
/*
 * curl -k --cert certs/client.pl2 --pass abc123 --cert-type p12 https://localhost:<PORT>
 */
/*
 * Certificate generation:
 */
/*
 * ## CA
 * certtool --generate-privkey --outfile ca.key
 * echo 'cn = GnuTLS test CA' > ca.tmpl
 * echo 'ca' >> ca.tmpl
 * echo 'cert_signing_key' >> ca.tmpl
 * echo 'expiration_days = 3650' >> ca.tmpl
 * certtool --generate-self-signed --load-privkey ca.key --template ca.tmpl --outfile ca.pem
 */
/*
 * ## Server
 * certtool --generate-privkey --outfile server.key
 * echo 'organization = GnuTLS test server' > server.tmpl
 * echo 'cn = test.gnutls.org' >> server.tmpl
 * echo 'tls_www_server' >> server.tmpl
 * echo 'expiration_days = 3650' >> server.tmpl
 * certtool --generate-certificate --load-ca-privkey ca.key --load-ca-certificate ca.pem --load-privkey
    server.key --template server.tmpl --outfile server.pem
 */
/*
 * ## Client
 * certtool --generate-privkey --outfile client.key
 * echo 'cn = GnuTLS test client' > client.tmpl
 * echo 'tls_www_client' >> client.tmpl
 * echo 'expiration_days = 3650' >> client.tmpl
 * certtool --generate-certificate --load-ca-certificate ca.pem --load-ca-privkey ca.key --load-privkey
    client.key --template client.tmpl --outfile client.pem
 * certtool --to-p12 --p12-name=MyKey --password=abc123 --load-ca-certificate ca.pem --load-privkey
    client.key --load-certificate client.pem --outder --outfile client.p12
 */

#define KEY_FILE SG_EXAMPLES_CERTS_DIR "/server.key"
#define CERT_FILE SG_EXAMPLES_CERTS_DIR "/server.pem"
#define CA_FILE SG_EXAMPLES_CERTS_DIR "/ca.pem"

#define ERR_SIZE 256
#define PAGE_FMT "<html><head><title>Hello world</title></head><body><font color=\"%s\">%s</font></font></body></html>"
#define SECRET_MSG "Secret"

static void concat(char *s1, ...) {
    va_list ap;
    const char *s;
    va_start(ap, s1);
    while ((s = va_arg(ap, const char *))
        strcat(s1, s);
    va_end(ap);
}

static bool sess_verify_cert(gnutls_session_t tls_session, const char *line_break, char *err) {
    gnutls_x509_crt_t cert = NULL;
    const gnutls_datum_t *certs;
    size_t len;
    unsigned int status, certs_size;
    int ret;
    if (!tls_session || !line_break || !err) {
        sg_strerror(EINVAL, err, ERR_SIZE);
        return false;
    }
}

```



```

    if ((ret = gnutls_certificate_verify_peers2(tls_session, &status)) != GNUTLS_E_SUCCESS) {
        concat(err, "Error verifying peers: ", gnutls_strerror(ret), line_break, NULL);
        goto fail;
    }
    if (status & GNUTLS_CERT_INVALID)
        concat(err, "The certificate is not trusted", line_break, NULL);
    if (status & GNUTLS_CERT_SIGNER_NOT_FOUND)
        concat(err, "The certificate has not got a known issuer", line_break, NULL);
    if (status & GNUTLS_CERT_REVOKED)
        concat(err, "The certificate has been revoked", line_break, NULL);
    if (gnutls_certificate_type_get(tls_session) != GNUTLS_CERT_X509) {
        concat(err, "The certificate type is not X.509", line_break, NULL);
        goto fail;
    }
    if ((ret = gnutls_x509_crt_init(&cert)) != GNUTLS_E_SUCCESS) {
        concat(err, "Error in the certificate initialization: ", gnutls_strerror(ret), line_break, NULL);
        goto fail;
    }
    if (!(certs = gnutls_certificate_get_peers(tls_session, &certs_size))) {
        concat(err, "No certificate was found", line_break, NULL);
        goto fail;
    }
    if ((ret = gnutls_x509_crt_import(cert, &certs[0], GNUTLS_X509_FMT_DER)) != GNUTLS_E_SUCCESS) {
        concat(err, "Error parsing certificate: ", gnutls_strerror(ret), line_break, NULL);
        goto fail;
    }
    if (gnutls_x509_crt_get_expiration_time(cert) < time(NULL)) {
        concat(err, "The certificate has expired", line_break, NULL);
        goto fail;
    }
    if (gnutls_x509_crt_get_activation_time(cert) > time(NULL)) {
        concat(err, "The certificate has not been activated yet", line_break, NULL);
        goto fail;
    }
}
fail:
    len = strlen(err);
    err[len - strlen("<br>")] = '\0';
    gnutls_x509_crt_deinit(cert);
    return len == 0;
}

static void req_cb(__SG_UNUSED void *cls, __SG_UNUSED struct sg_httpreq *req, struct
    sg_httpres *res) {
    char msg[ERR_SIZE];
    char *color, *page;
    size_t page_size;
    unsigned int status;
    if (sess_verify_cert(sg_httpreq_tls_session(req), "<br>", msg)) {
        strcpy(msg, SECRET_MSG);
        color = "green";
        status = 200;
    } else {
        color = "red";
        status = 500;
    }
    page_size = (size_t) snprintf(NULL, 0, PAGE_FMT, color, msg);
    page = sg_alloc(page_size);
    snprintf(page, page_size, PAGE_FMT, color, msg);
    sg_httpres_send(res, page, "text/html; charset=utf-8", status);
    sg_free(page);
}

int main(void) {
    struct sg_httpsrv *srv = sg_httpsrv_new(req_cb, NULL);
    gnutls_datum_t key_file, cert_file, ca_file;
    int ret, status = EXIT_FAILURE;
    memset(&key_file, 0, sizeof(gnutls_datum_t));
    memset(&cert_file, 0, sizeof(gnutls_datum_t));
    memset(&ca_file, 0, sizeof(gnutls_datum_t));
    if ((ret = gnutls_load_file(KEY_FILE, &key_file)) != GNUTLS_E_SUCCESS) {
        fprintf(stderr, "Error loading the private key \"%s\": %s\n", KEY_FILE, gnutls_strerror(ret));
        fflush(stdout);
        goto fail;
    }
    if ((ret = gnutls_load_file(CERT_FILE, &cert_file)) != GNUTLS_E_SUCCESS) {
        fprintf(stderr, "Error loading the certificate \"%s\": %s\n", CERT_FILE, gnutls_strerror(ret));
        fflush(stdout);
        goto fail;
    }
    if ((ret = gnutls_load_file(CA_FILE, &ca_file)) != GNUTLS_E_SUCCESS) {
        fprintf(stderr, "Error loading the CA \"%s\": %s\n", CA_FILE, gnutls_strerror(ret));
        fflush(stdout);
        goto fail;
    }
    if (sg_httpsrv_tls_listen2(srv, (const char *) key_file.data, NULL, (const char
        *) cert_file.data,
                                (const char *) ca_file.data, NULL, 0 /* 0 = port chosen randomly */, false))

```

```

    {
        status = EXIT_SUCCESS;
        fprintf(stdout, "Server running at https://localhost:%d\n",
            sg_httpsrv_port(srv));
        fflush(stdout);
        getchar();
    }
fail:
    sg_httpsrv_free(srv);
    if (key_file.size > 0)
        gnutls_free(key_file.data);
    if (cert_file.size > 0)
        gnutls_free(cert_file.data);
    if (ca_file.size > 0)
        gnutls_free(ca_file.data);
    return status;
}

```

8.8 example_httpuplds.c

Simple example showing how to upload files to the server.

```

/*
 *      _
 *   _/_/_/_/_/_/_/_/_/_/_
 *  /_/_/_/_/_/_/_/_/_/_/_
 * |_/_/_/_/_/_/_/_/_/_/_|
 * |_/_/_/_/_/_/_/_/_/_/_|
 *
 * -- an ideal C library to develop cross-platform HTTP servers.
 *
 * Copyright (c) 2016-2018 Silvio Clecio <silvioprog@gmail.com>
 *
 * This file is part of Sagui library.
 *
 * Sagui library is free software: you can redistribute it and/or modify
 * it under the terms of the GNU Lesser General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * Sagui library is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU Lesser General Public License for more details.
 *
 * You should have received a copy of the GNU Lesser General Public License
 * along with Sagui library. If not, see <http://www.gnu.org/licenses/>.
 */

#include <stdio.h>
#include <stdlib.h>
#include <limits.h>
#include <sagui.h>

/* NOTE: Error checking has been omitted to make it clear. */

#ifdef _WIN32
#define PATH_SEP '\\\\'
#else
#define PATH_SEP '/'
#endif

#define PAGE_FORM
    "<html>"
    "<body>"
    "<form action=\"\" method=\"post\" enctype=\"multipart/form-data\">"
    "<fieldset>"
    "<legend>Choose the files:</legend>"
    "File 1: <input type=\"file\" name=\"file1\"/><br>"
    "File 2: <input type=\"file\" name=\"file2\"/><br>"
    "<input type=\"submit\"/>"
    "</fieldset>"
    "</form>"
    "</body>"
    "</html>"

#define PAGE_DONE
    "<html>"
    "<head>"
    "<title>Uploads</title>"
    "</head>"

```



```

* Copyright (c) 2016-2018 Silvio Clecio <silvioprog@gmail.com>
*
* This file is part of Sagui library.
*
* Sagui library is free software: you can redistribute it and/or modify
* it under the terms of the GNU Lesser General Public License as published by
* the Free Software Foundation, either version 3 of the License, or
* (at your option) any later version.
*
* Sagui library is distributed in the hope that it will be useful,
* but WITHOUT ANY WARRANTY; without even the implied warranty of
* MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
* GNU Lesser General Public License for more details.
*
* You should have received a copy of the GNU Lesser General Public License
* along with Sagui library. If not, see <http://www.gnu.org/licenses/>.
*/

#include <stdlib.h>
#include <stdio.h>
#include <sagui.h>

/* NOTE: Error checking has been omitted to make it clear. */

static int get_segments_cb(__SG_UNUSED void *cls, const char *segment) {
    fprintf(stdout, " %s\n", segment);
    return 0;
}

static void route_cb(void *cls, struct sg_route *route) {
    fprintf(stdout, "%s: %s\n", sg_route_path(route), (const char *) cls);
    sg_route_get_segments(route, get_segments_cb, NULL);
}

int main(void) {
    struct sg_router *router;
    struct sg_route *routes = NULL;
    sg_routes_add(&routes, "/foo/[0-9]+", route_cb, "foo-data");
    sg_routes_add(&routes, "/bar/([a-zA-Z]+)", route_cb, "bar-data");
    router = sg_router_new(routes);
    sg_router_dispatch(router, "/foo/123", NULL);
    fprintf(stdout, "---\n");
    sg_router_dispatch(router, "/bar/abc", NULL);
    sg_routes_cleanup(&routes);
    sg_router_free(router);
    fflush(stdout);
    return EXIT_SUCCESS;
}

```

8.10 example_router_simple.c

Simple example showing how to use the router feature.

```

/*
 *
 * / _ | / _ | / _ | _ | _ | _ |
 * \ _ | \ _ | \ _ | _ | _ | _ |
 * | _ | \ _ | \ _ | \ _ | _ | _ |
 *
 *
 * -- an ideal C library to develop cross-platform HTTP servers.
 *
 * Copyright (c) 2016-2018 Silvio Clecio <silvioprog@gmail.com>
 *
 * This file is part of Sagui library.
 *
 * Sagui library is free software: you can redistribute it and/or modify
 * it under the terms of the GNU Lesser General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * Sagui library is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU Lesser General Public License for more details.
 *
 * You should have received a copy of the GNU Lesser General Public License
 * along with Sagui library. If not, see <http://www.gnu.org/licenses/>.
 */

#include <stdlib.h>

```



```

    return 0;
}

static void route_home_cb(__SG_UNUSED void *cls, struct sg_route *route) {
    struct holder *h = sg_route_user_data(route);
    sg_https_send(h->res, "<html><head><title>Home</title></head><body>Home</body></html>"
        , "text/html", 200);
}

static void route_download_cb(__SG_UNUSED void *cls, struct sg_route *route) {
    struct holder *h = sg_route_user_data(route);
    struct sg_str *page = sg_str_new();
    char file[256];
    memset(file, 0, sizeof(file));
    sg_route_get_vars(route, route_download_file_cb, file);
    if (strlen(file) == 0)
        strcpy(file, "Download");
    sg_str_printf(page, "<html><head><title>Download</title></head><body>%s</body></html>",
        file);
    sg_https_send(h->res, sg_str_content(page), "text/html", 200);
    sg_str_free(page);
}

static void route_about_cb(__SG_UNUSED void *cls, struct sg_route *route) {
    struct holder *h = sg_route_user_data(route);
    sg_https_send(h->res, "
        <html><head><title>About</title></head><body>About</body></html>", "text/html", 200);
}

static void req_cb(__SG_UNUSED void *cls, struct sg_httpreq *req, struct
    sg_https *res) {
    struct sg_router *router = cls;
    struct holder h = {req, res};
    if (sg_router_dispatch(router, sg_httpreq_path(req), &h) != 0)
        sg_https_send(res, "<html><head><title>Not
            found</title></head><body>404</body></html>", "text/html", 404);
}

int main(void) {
    struct sg_route *routes = NULL;
    struct sg_router *router;
    struct sg_httpsrv *srv;
    sg_routes_add(&routes, "/home", route_home_cb, NULL);
    sg_routes_add(&routes, "/download", route_download_cb, NULL);
    sg_routes_add(&routes, "/download/(?P<file>[a-z]+)", route_download_cb, NULL);
    sg_routes_add(&routes, "/about", route_about_cb, NULL);
    router = sg_router_new(routes);
    srv = sg_httpsrv_new(req_cb, router);
    if (!sg_httpsrv_listen(srv, 0 /* 0 = port chosen randomly */, false)) {
        sg_httpsrv_free(srv);
        return EXIT_FAILURE;
    }
    fprintf(stdout, "Server running at http://localhost:%d\n", sg_httpsrv_port(srv));
    fflush(stdout);
    getchar();
    sg_httpsrv_free(srv);
    sg_routes_cleanup(&routes);
    sg_router_free(router);
    return EXIT_SUCCESS;
}

```

8.12 example router vars.c

Simple example showing how to access the path variables of the router feature.

```
/*  
 *      _  
 *   _/_/_/_/_/_/_/  
 *  /_/_/_/_/_/_/_/  
 * |_/___|_|_|_|_|_  
 * |_/_\__|_|_|_|_|_  
 *       |_|_|_|_|_  
 *  
 * -- an ideal C library to develop cross-platform HTTP servers.  
 *  
 * Copyright (c) 2016-2018 Silvio Clecio <silvioprog@gmail.com>  
 *  
 * This file is part of Sagui library.  
 *  
 * Sagui library is free software: you can redistribute it and/or modify  
 * it under the terms of the GNU Lesser General Public License as published by  
 * the Free Software Foundation, either version 3 of the License, or
```

```

* (at your option) any later version.
*
* Sagui library is distributed in the hope that it will be useful,
* but WITHOUT ANY WARRANTY; without even the implied warranty of
* MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
* GNU Lesser General Public License for more details.
*
* You should have received a copy of the GNU Lesser General Public License
* along with Sagui library. If not, see <http://www.gnu.org/licenses/>.
*/

#include <stdlib.h>
#include <stdio.h>
#include <sagui.h>

/* NOTE: Error checking has been omitted to make it clear. */

static int get_vars_cb(__SG_UNUSED void *cls, const char *name, const char *val) {
    fprintf(stdout, " %s: %s\n", name, val);
    return 0;
}

static void route_cb(void *cls, struct sg_route *route) {
    fprintf(stdout, "%s: %s\n", sg_route_path(route), (const char *) cls);
    sg_route_get_vars(route, get_vars_cb, NULL);
}

int main(void) {
    struct sg_router *router;
    struct sg_route *routes = NULL;
    sg_routes_add(&routes, "/foo/bar", route_cb, "foo-bar-data");
    sg_routes_add(&routes, "/bar", route_cb, "bar-data");
    sg_routes_add(&routes, "/customer/(?P<name>[a-zA-Z]+)", route_cb, "customer-data");
    sg_routes_add(&routes, "/product/(?P<id>[0-9]+)", route_cb, "product-data");
    sg_routes_add(&routes, "/employee/(?P<id>[0-9]+)/[a|i]", route_cb, "employee-data");
    router = sg_router_new(routes);
    sg_router_dispatch(router, "/foo/bar", NULL);
    fprintf(stdout, "---\n");
    sg_router_dispatch(router, "/customer/Torvalds", NULL);
    fprintf(stdout, "---\n");
    sg_router_dispatch(router, "/product/123", NULL);
    fprintf(stdout, "---\n");
    sg_router_dispatch(router, "/employee/123/i", NULL);
    sg_routes_cleanup(&routes);
    sg_router_free(router);
    fflush(stdout);
    return EXIT_SUCCESS;
}

```

8.13 example_str.c

Simple example showing the `sg_str` feature.

```

/*
 *
 * / _ _ | / _ _ | / _ _ | / _ _ | / _ _ |
 * \ _ _ | \ _ _ | \ _ _ | \ _ _ | \ _ _ |
 * | _ _ | | _ _ | | _ _ | | _ _ | | _ _ |
 *
 * -- an ideal C library to develop cross-platform HTTP servers.
 *
 * Copyright (c) 2016-2018 Silvio Clecio <silvioprog@gmail.com>
 *
 * This file is part of Sagui library.
 *
 * Sagui library is free software: you can redistribute it and/or modify
 * it under the terms of the GNU Lesser General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * Sagui library is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU Lesser General Public License for more details.
 *
 * You should have received a copy of the GNU Lesser General Public License
 * along with Sagui library. If not, see <http://www.gnu.org/licenses/>.
 */

/* NOTE: Error checking has been omitted to make it clear. */

```

```

#include <stdio.h>
#include <stdlib.h>
#include <sagui.h>

int main(void) {
    struct sg_str *str = sg_str_new();
    sg_str_printf(str, "%s %s", "Hello", "world");
    printf("%s", sg_str_content(str));
    sg_str_free(str);
    return EXIT_SUCCESS;
}

```

8.14 example_strmap.c

Simple example showing the `sg_strmap` feature.

```

/*
 *
 *  / _ | / _ | / _ | / _ |
 *  \ _ | \ _ | \ _ | \ _ |
 *  | _ | | _ | | _ | | _ |
 *  | _ | | _ | | _ | | _ |
 *
 *
 * -- an ideal C library to develop cross-platform HTTP servers.
 *
 * Copyright (c) 2016-2018 Silvio Clecio <silvioprog@gmail.com>
 *
 * This file is part of Sagui library.
 *
 * Sagui library is free software: you can redistribute it and/or modify
 * it under the terms of the GNU Lesser General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * Sagui library is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU Lesser General Public License for more details.
 *
 * You should have received a copy of the GNU Lesser General Public License
 * along with Sagui library. If not, see <http://www.gnu.org/licenses/>.
 */

/* NOTE: Error checking has been omitted to make it clear. */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sagui.h>

static int map_sort(__SG_UNUSED void *cls, struct sg_strmap *pair_a, struct
    sg_strmap *pair_b) {
    return strcmp(sg_strmap_val(pair_b), sg_strmap_val(pair_a)); /* desc */
}

static int map_iter(__SG_UNUSED void *cls, struct sg_strmap *pair) {
    const char *name = sg_strmap_name(pair);
    printf("\t%c: %s\n", *name, name);
    return 0;
}

static void chat(struct sg_strmap **map, const char *name, const char *msg) {
    struct sg_strmap *pair;
    sg_strmap_set(map, name, msg);
    if (msg && (sg_strmap_find(*map, name, &pair) == 0))
        printf("%c:\t%s\n", *sg_strmap_name(pair), sg_strmap_val(pair));
}

int main(void) {
    struct sg_strmap *map = NULL;
    chat(&map, "Clecio", "Hello!");
    chat(&map, "Paim", "Hello. How are you?");
    chat(&map, "Clecio", "I'm fine. And you?");
    chat(&map, "Paim", "Me too.");
    printf("\nChatters:\n");
    sg_strmap_sort(&map, &map_sort, NULL);
    sg_strmap_iter(map, &map_iter, NULL);
    sg_strmap_cleanup(&map);
    return EXIT_SUCCESS;
}

```


Index

API reference, 3

example_entrypoint.h, 82
example_httppauth.h, 82
example_httpcookie.h, 82
example_httpsrv.h, 82
example_httpsrv_benchmark.h, 82
example_httpsrv_tls.h, 82
example_httpsrv_tls_cert_auth.h, 82
example_httpuplds.h, 82
example_router_segments.h, 82
example_router_simple.h, 82
example_router_srv.h, 82
example_router_vars.h, 82
example_str.h, 82
example_strmap.h, 82

HTTP server, 24

sg_httppauth_cancel, 29
sg_httppauth_cb, 26
sg_httppauth_deny, 29
sg_httppauth_pwd, 30
sg_httppauth_realm, 28
sg_httppauth_set_realm, 28
sg_httppauth_usr, 30
sg_httpread_end, 58
sg_httpreq_cb, 27
sg_httpreq_cookies, 37
sg_httpreq_fields, 38
sg_httpreq_headers, 36
sg_httpreq_is_uploading, 40
sg_httpreq_method, 39
sg_httpreq_params, 37
sg_httpreq_path, 39
sg_httpreq_payload, 40
sg_httpreq_set_user_data, 42
sg_httpreq_tls_session, 41
sg_httpreq_uploads, 41
sg_httpreq_user_data, 42
sg_httpreq_version, 38
sg_httpres_clear, 46
sg_httpres_headers, 43
sg_httpres_send, 25
sg_httpres_sendbinary, 44
sg_httpres_sendfile, 44
sg_httpres_sendstream, 45
sg_httpres_set_cookie, 43
sg_httpsrv_con_limit, 57
sg_httpsrv_con_timeout, 56
sg_httpsrv_free, 48
sg_httpsrv_is_threaded, 51
sg_httpsrv_listen, 49
sg_httpsrv_new, 47
sg_httpsrv_new2, 46
sg_httpsrv_payld_limit, 54
sg_httpsrv_port, 50

sg_httpsrv_post_buf_size, 53
sg_httpsrv_set_con_limit, 57
sg_httpsrv_set_con_timeout, 56
sg_httpsrv_set_payld_limit, 53
sg_httpsrv_set_post_buf_size, 52
sg_httpsrv_set_thr_pool_size, 55
sg_httpsrv_set_upld_cbs, 51
sg_httpsrv_set_upld_dir, 52
sg_httpsrv_set_uplds_limit, 54
sg_httpsrv_shutdown, 50
sg_httpsrv_thr_pool_size, 55
sg_httpsrv_tls_listen, 49
sg_httpsrv_tls_listen2, 48
sg_httpsrv_upld_dir, 52
sg_httpsrv_uplds_limit, 55
sg_httpupld_cb, 27
sg_httpupld_dir, 33
sg_httpupld_encoding, 34
sg_httpupld_field, 33
sg_httpupld_handle, 32
sg_httpupld_mime, 34
sg_httpupld_name, 33
sg_httpupld_save, 35
sg_httpupld_save_as, 36
sg_httpupld_size, 35
sg_httpuplds_count, 32
sg_httpuplds_iter, 31
sg_httpuplds_iter_cb, 27
sg_httpuplds_next, 31

Path routing, 59

sg_entrypoint_name, 62
sg_entrypoint_set_user_data, 63
sg_entrypoint_user_data, 63
sg_entrypoints_add, 64
sg_entrypoints_clear, 66
sg_entrypoints_find, 66
sg_entrypoints_free, 64
sg_entrypoints_iter, 65
sg_entrypoints_iter_cb, 60
sg_entrypoints_new, 63
sg_entrypoints_rm, 65
sg_get_segments_cb, 60
sg_get_vars_cb, 60
sg_route_cb, 61
sg_route_get_segments, 69
sg_route_get_vars, 70
sg_route_handle, 67
sg_route_match, 67
sg_route_path, 69
sg_route_pattern, 68
sg_route_rawpattern, 68
sg_route_user_data, 70
sg_router_dispatch, 76
sg_router_dispatch2, 75

- sg_router_dispatch_cb, 61
- sg_router_free, 75
- sg_router_match_cb, 62
- sg_router_new, 74
- sg_routes_add, 72
- sg_routes_add2, 71
- sg_routes_cleanup, 74
- sg_routes_count, 74
- sg_routes_iter, 73
- sg_routes_iter_cb, 61
- sg_routes_next, 73
- sg_routes_rm, 72
- SG_ERR_SIZE
 - sagui.h, 85
- sagui.h, 82
 - SG_ERR_SIZE, 85
- sg_alloc
 - Utilities, 7
- sg_entrpoint, 78
- sg_entrpoint_name
 - Path routing, 62
- sg_entrpoint_set_user_data
 - Path routing, 63
- sg_entrpoint_user_data
 - Path routing, 63
- sg_entrpoints, 78
- sg_entrpoints_add
 - Path routing, 64
- sg_entrpoints_clear
 - Path routing, 66
- sg_entrpoints_find
 - Path routing, 66
- sg_entrpoints_free
 - Path routing, 64
- sg_entrpoints_iter
 - Path routing, 65
- sg_entrpoints_iter_cb
 - Path routing, 60
- sg_entrpoints_new
 - Path routing, 63
- sg_entrpoints_rm
 - Path routing, 65
- sg_err_cb
 - Utilities, 4
- sg_extract_entrpoint
 - Utilities, 9
- sg_free
 - Utilities, 8
- sg_free_cb
 - Utilities, 5
- sg_get_segments_cb
 - Path routing, 60
- sg_get_vars_cb
 - Path routing, 60
- sg_httpauth, 78
- sg_httpauth_cancel
 - HTTP server, 29
- sg_httpauth_cb
 - HTTP server, 26
- sg_httpauth_deny
 - HTTP server, 29
- sg_httpauth_pwd
 - HTTP server, 30
- sg_httpauth_realm
 - HTTP server, 28
- sg_httpauth_set_realm
 - HTTP server, 28
- sg_httpauth_usr
 - HTTP server, 30
- sg_httpread_end
 - HTTP server, 58
- sg_httpreq, 79
- sg_httpreq_cb
 - HTTP server, 27
- sg_httpreq_cookies
 - HTTP server, 37
- sg_httpreq_fields
 - HTTP server, 38
- sg_httpreq_headers
 - HTTP server, 36
- sg_httpreq_is_uploading
 - HTTP server, 40
- sg_httpreq_method
 - HTTP server, 39
- sg_httpreq_params
 - HTTP server, 37
- sg_httpreq_path
 - HTTP server, 39
- sg_httpreq_payload
 - HTTP server, 40
- sg_httpreq_set_user_data
 - HTTP server, 42
- sg_httpreq_tls_session
 - HTTP server, 41
- sg_httpreq_uploads
 - HTTP server, 41
- sg_httpreq_user_data
 - HTTP server, 42
- sg_httpreq_version
 - HTTP server, 38
- sg_https, 79
- sg_https_clear
 - HTTP server, 46
- sg_https_headers
 - HTTP server, 43
- sg_https_send
 - HTTP server, 25
- sg_https_sendbinary
 - HTTP server, 44
- sg_https_sendfile
 - HTTP server, 44
- sg_https_sendstream
 - HTTP server, 45
- sg_https_set_cookie
 - HTTP server, 43
- sg_httpsrv, 79

- sg_httpsrv_con_limit
 - HTTP server, [57](#)
- sg_httpsrv_con_timeout
 - HTTP server, [56](#)
- sg_httpsrv_free
 - HTTP server, [48](#)
- sg_httpsrv_is_threaded
 - HTTP server, [51](#)
- sg_httpsrv_listen
 - HTTP server, [49](#)
- sg_httpsrv_new
 - HTTP server, [47](#)
- sg_httpsrv_new2
 - HTTP server, [46](#)
- sg_httpsrv_payld_limit
 - HTTP server, [54](#)
- sg_httpsrv_port
 - HTTP server, [50](#)
- sg_httpsrv_post_buf_size
 - HTTP server, [53](#)
- sg_httpsrv_set_con_limit
 - HTTP server, [57](#)
- sg_httpsrv_set_con_timeout
 - HTTP server, [56](#)
- sg_httpsrv_set_payld_limit
 - HTTP server, [53](#)
- sg_httpsrv_set_post_buf_size
 - HTTP server, [52](#)
- sg_httpsrv_set_thr_pool_size
 - HTTP server, [55](#)
- sg_httpsrv_set_upld_cbs
 - HTTP server, [51](#)
- sg_httpsrv_set_upld_dir
 - HTTP server, [52](#)
- sg_httpsrv_set_uplds_limit
 - HTTP server, [54](#)
- sg_httpsrv_shutdown
 - HTTP server, [50](#)
- sg_httpsrv_thr_pool_size
 - HTTP server, [55](#)
- sg_httpsrv_tls_listen
 - HTTP server, [49](#)
- sg_httpsrv_tls_listen2
 - HTTP server, [48](#)
- sg_httpsrv_upld_dir
 - HTTP server, [52](#)
- sg_httpsrv_uplds_limit
 - HTTP server, [55](#)
- sg_httpupld, [80](#)
- sg_httpupld_cb
 - HTTP server, [27](#)
- sg_httpupld_dir
 - HTTP server, [33](#)
- sg_httpupld_encoding
 - HTTP server, [34](#)
- sg_httpupld_field
 - HTTP server, [33](#)
- sg_httpupld_handle
 - HTTP server, [32](#)
- sg_httpupld_mime
 - HTTP server, [34](#)
- sg_httpupld_name
 - HTTP server, [33](#)
- sg_httpupld_save
 - HTTP server, [35](#)
- sg_httpupld_save_as
 - HTTP server, [36](#)
- sg_httpupld_size
 - HTTP server, [35](#)
- sg_httpuplds_count
 - HTTP server, [32](#)
- sg_httpuplds_iter
 - HTTP server, [31](#)
- sg_httpuplds_iter_cb
 - HTTP server, [27](#)
- sg_httpuplds_next
 - HTTP server, [31](#)
- sg_is_post
 - Utilities, [8](#)
- sg_read_cb
 - Utilities, [5](#)
- sg_realloc
 - Utilities, [7](#)
- sg_route, [80](#)
- sg_route_cb
 - Path routing, [61](#)
- sg_route_get_segments
 - Path routing, [69](#)
- sg_route_get_vars
 - Path routing, [70](#)
- sg_route_handle
 - Path routing, [67](#)
- sg_route_match
 - Path routing, [67](#)
- sg_route_path
 - Path routing, [69](#)
- sg_route_pattern
 - Path routing, [68](#)
- sg_route_rawpattern
 - Path routing, [68](#)
- sg_route_user_data
 - Path routing, [70](#)
- sg_router, [80](#)
- sg_router_dispatch
 - Path routing, [76](#)
- sg_router_dispatch2
 - Path routing, [75](#)
- sg_router_dispatch_cb
 - Path routing, [61](#)
- sg_router_free
 - Path routing, [75](#)
- sg_router_match_cb
 - Path routing, [62](#)
- sg_router_new
 - Path routing, [74](#)
- sg_routes_add

- Path routing, [72](#)
- sg_routes_add2
 - Path routing, [71](#)
- sg_routes_cleanup
 - Path routing, [74](#)
- sg_routes_count
 - Path routing, [74](#)
- sg_routes_iter
 - Path routing, [73](#)
- sg_routes_iter_cb
 - Path routing, [61](#)
- sg_routes_next
 - Path routing, [73](#)
- sg_routes_rm
 - Path routing, [72](#)
- sg_save_as_cb
 - Utilities, [6](#)
- sg_save_cb
 - Utilities, [5](#)
- sg_str, [81](#)
- sg_str_clear
 - String, [13](#)
- sg_str_content
 - String, [12](#)
- sg_str_free
 - String, [10](#)
- sg_str_length
 - String, [13](#)
- sg_str_new
 - String, [10](#)
- sg_str_printf
 - String, [12](#)
- sg_str_printf_va
 - String, [11](#)
- sg_str_write
 - String, [11](#)
- sg_strerror
 - Utilities, [8](#)
- sg_strmap, [81](#)
- sg_strmap_add
 - String map, [17](#)
- sg_strmap_cleanup
 - String map, [23](#)
- sg_strmap_count
 - String map, [22](#)
- sg_strmap_find
 - String map, [18](#)
- sg_strmap_get
 - String map, [19](#)
- sg_strmap_iter
 - String map, [21](#)
- sg_strmap_iter_cb
 - String map, [15](#)
- sg_strmap_name
 - String map, [16](#)
- sg_strmap_next
 - String map, [22](#)
- sg_strmap_rm
- String map, [19](#)
- sg_strmap_set
 - String map, [18](#)
- sg_strmap_sort
 - String map, [21](#)
- sg_strmap_sort_cb
 - String map, [16](#)
- sg_strmap_val
 - String map, [17](#)
- sg_tmpdir
 - Utilities, [9](#)
- sg_version
 - Utilities, [6](#)
- sg_version_str
 - Utilities, [6](#)
- sg_write_cb
 - Utilities, [4](#)
- String, [10](#)
 - sg_str_clear, [13](#)
 - sg_str_content, [12](#)
 - sg_str_free, [10](#)
 - sg_str_length, [13](#)
 - sg_str_new, [10](#)
 - sg_str_printf, [12](#)
 - sg_str_printf_va, [11](#)
 - sg_str_write, [11](#)
- String map, [15](#)
 - sg_strmap_add, [17](#)
 - sg_strmap_cleanup, [23](#)
 - sg_strmap_count, [22](#)
 - sg_strmap_find, [18](#)
 - sg_strmap_get, [19](#)
 - sg_strmap_iter, [21](#)
 - sg_strmap_iter_cb, [15](#)
 - sg_strmap_name, [16](#)
 - sg_strmap_next, [22](#)
 - sg_strmap_rm, [19](#)
 - sg_strmap_set, [18](#)
 - sg_strmap_sort, [21](#)
 - sg_strmap_sort_cb, [16](#)
 - sg_strmap_val, [17](#)
- Utilities, [4](#)
 - sg_alloc, [7](#)
 - sg_err_cb, [4](#)
 - sg_extract_entrpoint, [9](#)
 - sg_free, [8](#)
 - sg_free_cb, [5](#)
 - sg_is_post, [8](#)
 - sg_read_cb, [5](#)
 - sg_realloc, [7](#)
 - sg_save_as_cb, [6](#)
 - sg_save_cb, [5](#)
 - sg_strerror, [8](#)
 - sg_tmpdir, [9](#)
 - sg_version, [6](#)
 - sg_version_str, [6](#)
 - sg_write_cb, [4](#)