ssh-ca-1.0

Generated by Doxygen 1.8.17

Chapter 1

SSH CA – Backend

1.1 Requirements

```
sudo apt install libcurl4-openssl-dev build-essential libssl-dev git
sudo apt install nginx fcgiwrap
sudo apt install doxygen doxygen-latex  graphviz
```

- A Redis Server. (Version: >=2.6.12)

1.2 Install

1. Go to /opt/ and clone this repo `git clone --recurse-submodules git@gitlab.gwdg.de:asta/dnd-referat/ssh-ca.git`.
2. Comment out all `location` blocks in your nginx site config and add this line: `include /opt/ssh-ca/site.conf;`
3. `sudo service nginx restart`

Tested on GWDG Cloud Servers Ubuntu 20.04 LTS.

1.3 Building

TODO (make all)

1.4 Configure

TODO

1.5 Code Documentation

The full backend/code documentation can be found here as HTML and here as PDF.

1.6 Coding Style

TODO

1.7 Technical Concepts

TODO

1.8 API Endpoints

TODO

Chapter 2

Data Structure Index

2.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 3

File Index

3.1 File List

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

Chapter 4

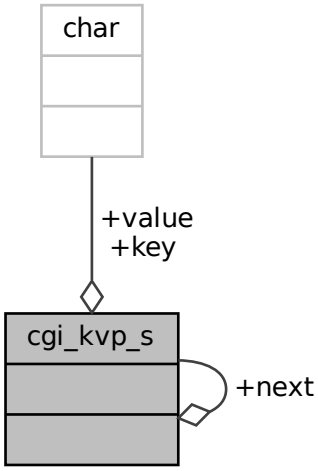
Data Structure Documentation

4.1 cgi_kvp_s Struct Reference

key-value-pairs struct

`#include <cgi.h>`

Collaboration diagram for cgi_kvp_s:



Data Fields

- char * key
  the key
- char * value
  the value associated with the key
- struct cgi_kvp_s * next
  the next pair

4.1.1 Detailed Description

key-value-pairs struct

Basically a linked list of key-value-pairs.

Definition at line 22 of file cgi.h.

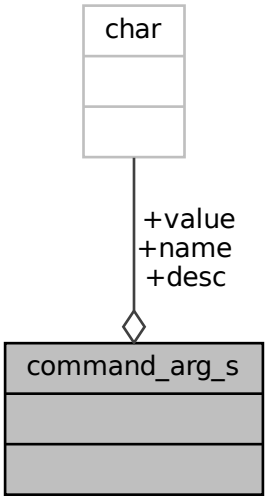
The documentation for this struct was generated from the following file:

- src/cgi.h

4.2 command_arg_s Struct Reference

command argument struct

Collaboration diagram for command_arg_s:



Data Fields

- char name [32]

  name of the argument
- char desc [128]

  description of the argument
- char value [MAXARGLEN+1]

  buffer for storing the value

4.2.1 Detailed Description

command argument struct

Definition at line 44 of file admin-util.c.

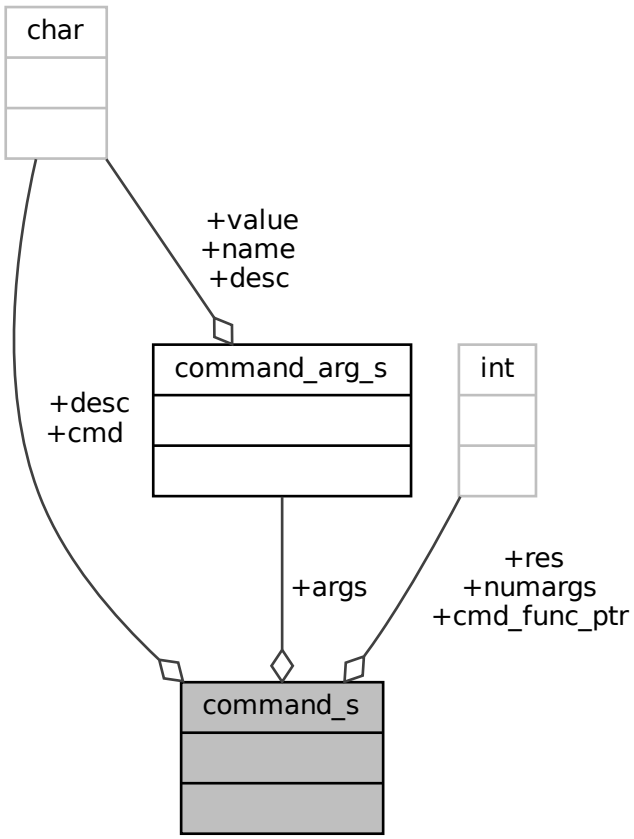
The documentation for this struct was generated from the following file:

- src/admin-util.c

4.3 command_s Struct Reference

command struct

Collaboration diagram for command_s:



Data Fields

- char cmd [32]

  name of the command

- char desc [128]

  description of the command

- int numargs

  number of arguments the command expects

- struct command_arg_s args [4]

  declaration of arguments

- int(* cmd_func_ptr)(char *errmsg, const char *cmd, const struct config_s *config, db_t *db, struct command_arg_s *args, int numargs)

  command function

- int res

  return value of command function

4.3.1 Detailed Description

command struct

Definition at line 54 of file admin-util.c.

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

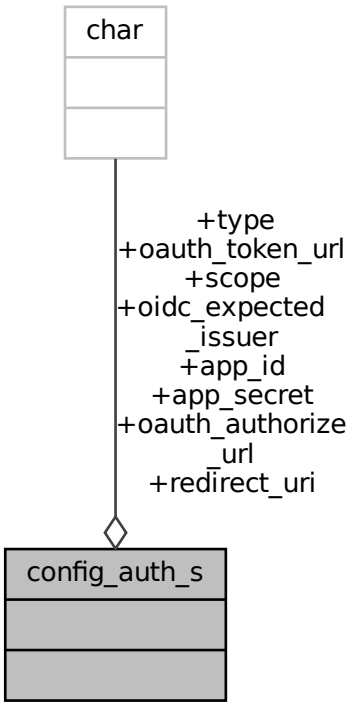
- src/admin-util.c

4.4 config_auth_s Struct Reference

native representation of config.auth

```
#include <config.h>
```

Collaboration diagram for config_auth_s:



Data Fields

- char * type

  authentcation type (currently only oidc (OpenID Connect) is supported)
- char * oauth_authorize_url

  the oauth authorize endpoint of the identity provider
- char * app_id

  oauth app id
- char * redirect_uri

  oauth redirect url
- char * scope

  oauth scope
- char * app_secret

  oauth scope
- char * oauth_token_url

  the oauth/oidc token endpoint of the identity provider
- char * oidc_expected_issuer

  the expected issuer (iss) claim in id_token

4.4.1 Detailed Description

native representation of config.auth

Definition at line 32 of file config.h.

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

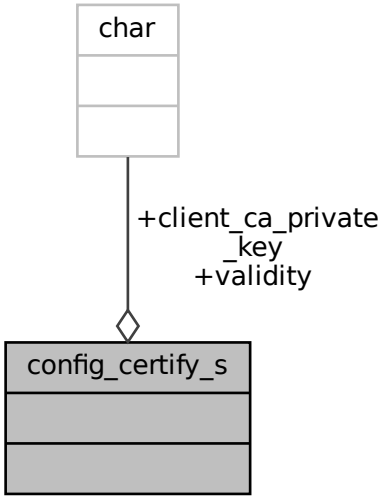
- src/config.h

4.5 config_certify_s Struct Reference

native representation of config["ssh-sign"]

```
#include <config.h>
```

Collaboration diagram for config_certify_s:



Data Fields

- char * client_ca_private_key

  path on the server to the private client-ca key
- char * validity

  how long a certified key should be valid for, see TIME FORMATS in sshd_config(5)

4.5.1 Detailed Description

native representation of config["ssh-sign"]

Definition at line 57 of file config.h.

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

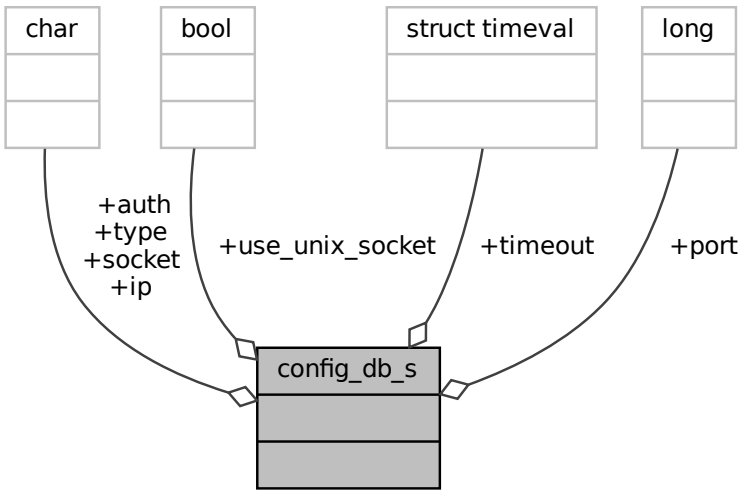
- src/config.h

4.6 config_db_s Struct Reference

native representation of config.db

`#include <config.h>`

Collaboration diagram for config_db_s:



Data Fields

- char * type
  database type (currently only redis supported)
- char * ip
  server ip
- long port
  server port
- char * socket
  server unix socket
- bool use_unix_socket
  whether to user the unix socket or ip+port
- struct timeval timeout
  redis timeout
- char * auth
  redis auth string, may be NULL if no authentication

4.6.1 Detailed Description

native representation of config.db

Definition at line 18 of file config.h.

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

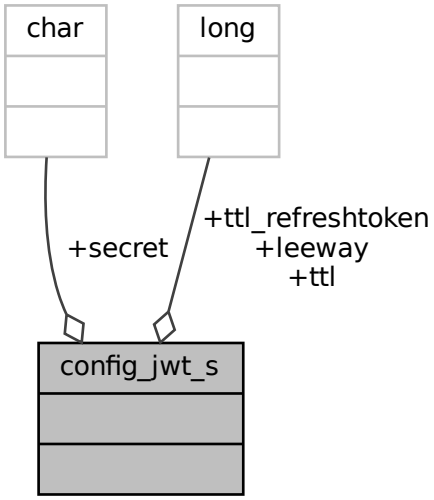
- src/config.h

4.7 config_jwt_s Struct Reference

native representation of config.jwt

`#include <config.h>`

Collaboration diagram for config_jwt_s:



Data Fields

- char * secret
  JWT secret.
- long ttl
  seconds until a JWT expires
- long leeway
  leeway to be added to ttl
- long ttl_refreshtoken
  seconds until a refreshtoken expires

4.7.1 Detailed Description

native representation of config.jwt

Definition at line 46 of file config.h.

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

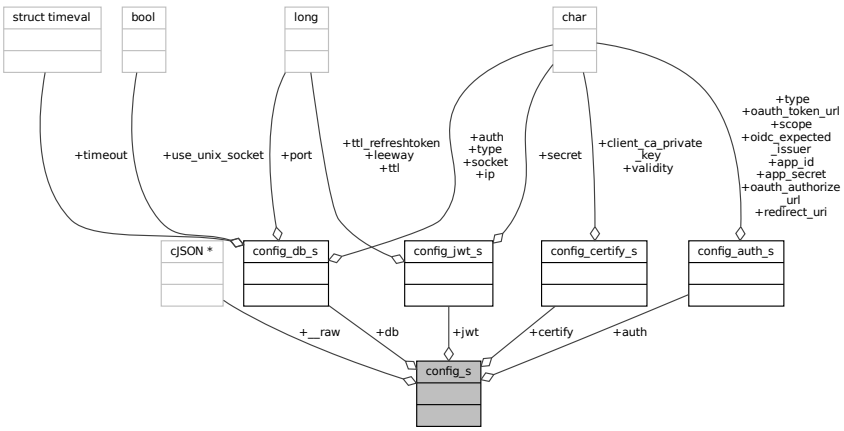
- src/config.h

4.8 config_s Struct Reference

config struct

```
#include <config.h>
```

Collaboration diagram for config_s:



Data Fields

- cJSON * __raw

  the raw json data (internal use only!)
- struct config_db_s db

  config.db
- struct config_auth_s auth

  config.auth
- struct config_jwt_s jwt

  config.jwt
- struct config_certify_s certify

  config["ssh-sign"]

4.8.1 Detailed Description

config struct

Definition at line 66 of file config.h.

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

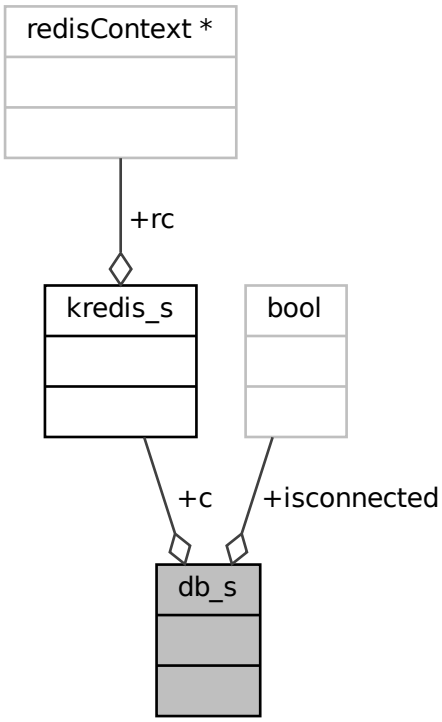
- src/config.h

4.9 db_s Struct Reference

database struct

`#include <db.h>`

Collaboration diagram for db_s:



Data Fields

- bool isconnected

  whether the kredis context is connected to a database
- kredis_t * c

  the kredis context

4.9.1 Detailed Description

database struct

Definition at line 42 of file db.h.

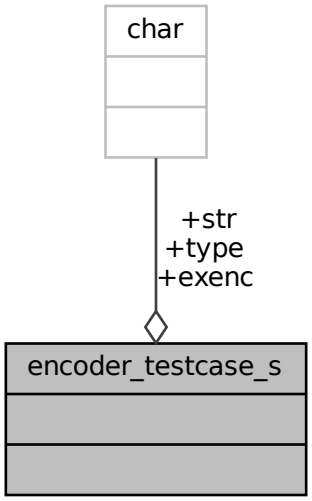
The documentation for this struct was generated from the following file:

- src/db.h

4.10 encoder_testcase_s Struct Reference

struct used for testing encoders

Collaboration diagram for encoder_testcase_s:



Data Fields

- char type

  '+': str is valid '-': str is invalid '\0': end of array
- char str [256]

  testcase string
- char exenc [256 ∗3+1]

  expected encoded of str

4.10.1 Detailed Description

struct used for testing encoders

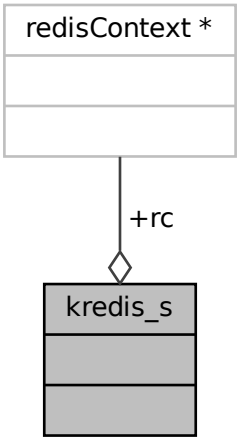
Definition at line 46 of file test-util.c.

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

- src/test-util.c

4.11 kredis_s Struct Reference

Collaboration diagram for kredis_s:



Data Fields

- redisContext * rc
  the redisContext from hiredis

4.11.1 Detailed Description

A kredis context struct

Definition at line 28 of file kredis.c.

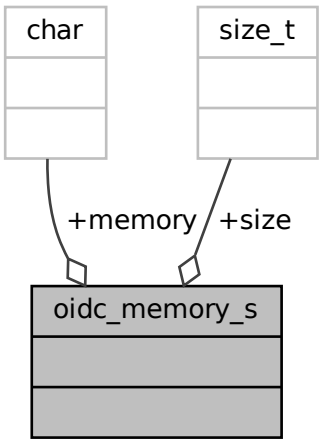
The documentation for this struct was generated from the following file:

- src/kredis.c

4.12 oidc_memory_s Struct Reference

internal struct used for receiving curl callback data

Collaboration diagram for oidc_memory_s:



Data Fields

- char * memory

  pointer to the allocated memory, containing the data
- size_t size

  size of memory

4.12.1 Detailed Description

internal struct used for receiving curl callback data

Definition at line 30 of file oidc.c.

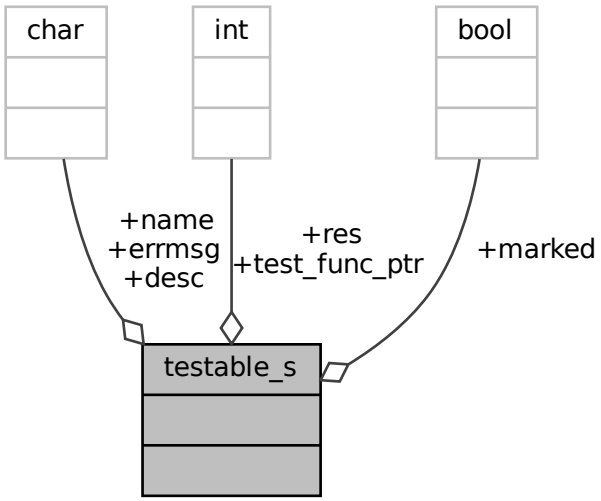
The documentation for this struct was generated from the following file:

- src/oidc.c

4.13 testable_s Struct Reference

testcase struct

Collaboration diagram for testable_s:



Data Fields

- char name [32]
  name of the test
- char desc [128]
  description of the test
- int(∗ test_func_ptr)(char ∗errmsg, const char ∗name, const struct config_s ∗config)
  test function
- bool marked
  true: should be tested false: should not be tested
- int res
  return value of test function. 0 indicated failure
- char errmsg [ERRMSGMAXSIZE]
  buffer for error message

4.13.1 Detailed Description

testcase struct

Definition at line 35 of file test-util.c.

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

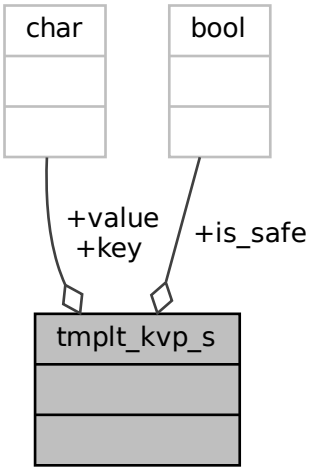
- src/test-util.c

4.14 tmplt_kvp_s Struct Reference

Template Key-Value-Pair Struct.

`#include <template_engine.h>`

Collaboration diagram for tmplt_kvp_s:



Data Fields

- const char * key
  the key
- const char * value
  the value that will replace %key%%
- bool is_safe
  whether the string is safe and can be used as is, or the string is unsafe and needs to be encoded

4.14.1 Detailed Description

Template Key-Value-Pair Struct.

Note

An Array of tmplt_kvp_s should be terminated by an element with key set to NULL.

Definition at line 33 of file template_engine.h.

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

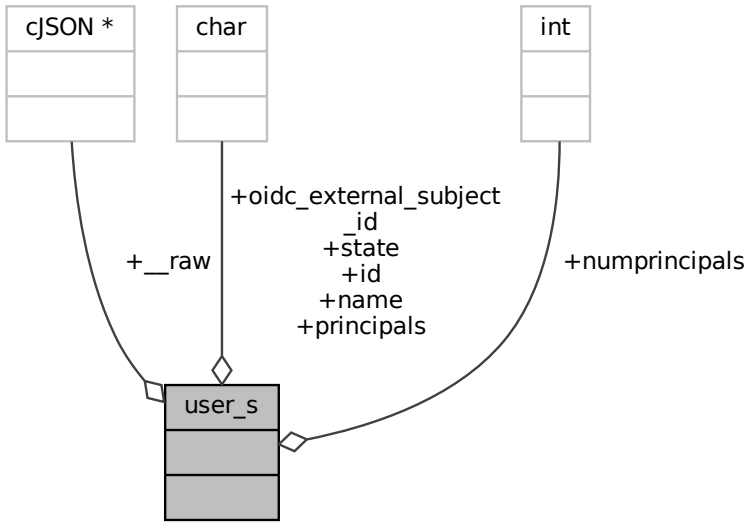
- src/template_engine.h

4.15 user_s Struct Reference

user struct

`#include <user.h>`

Collaboration diagram for user_s:



Data Fields

- cJSON * __raw
  the raw json data (internal use only!)
- char * id
  the userid
- char * name
  the name
- char * oidc_external_subject_id
  the users external subject id in the identity provider
- char * state
  the state, either "active" or "blocked"
- int numprincipals
  the number of principals in principals
- char ** principals
  null-terminated list of principal strings

4.15.1 Detailed Description

user struct

Definition at line 18 of file user.h.

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

- src/user.h

Chapter 5

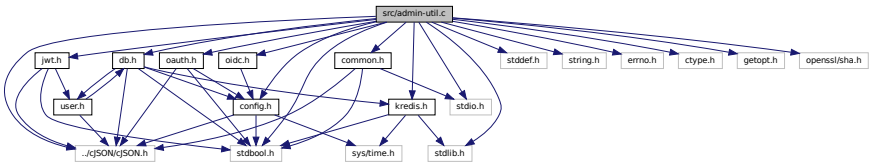
File Documentation

5.1 src/admin-util.c File Reference

Simple tool administration for administrating the SSH CA.

```
#include "common.h"
#include "db.h"
#include "oidc.h"
#include "oauth.h"
#include "kredis.h"
#include "jwt.h"
#include "config.h"
#include "../cJSON/cJSON.h"
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include <string.h>
#include <errno.h>
#include <ctype.h>
#include <getopt.h>
#include <openssl/sha.h>
```

Include dependency graph for admin-util.c:



Data Structures

- struct command_arg_s

  command argument struct
- struct command_s

  command struct

Macros

- #define MAXFILEPATHLEN 256

  the maximum length of a file path
- #define MAXARGLEN 256

  the maximum length of a command argument

Functions

- int main (int argc, char **argv, char **envp)

  main function for admin-util

5.1.1 Detailed Description

Simple tool administration for administrating the SSH CA.

Note

This tool will be removed as soon as a fully working REST API + Frontend exists.

See also

https://gitlab.gwdg.de/asta/dnd-referat/ssh-ca/-/issues/14

5.1.2 Function Documentation

5.1.2.1 main()

```
int main (
            int argc,
            char ** argv,
            char ** envp )
```

main function for admin-util

Parameters

in	*argc*	number of arguments from the command line
in	*argv*	arguments from the command line
in	*envp*	environment variables

Returns

the exit code of this program.

Return values

0	success
!0	failure

Definition at line 422 of file admin-util.c.

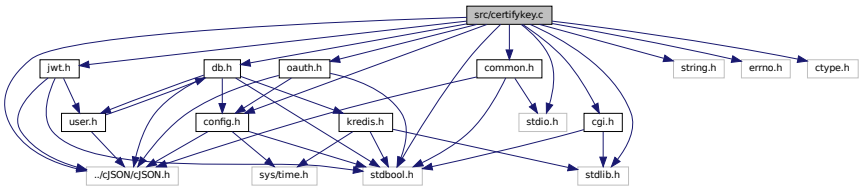
References config_s::db, ERRMSGMAXSIZE, MAXFILEPATHLEN, and command_s::res.

5.2 src/certifykey.c File Reference

key signing utility. handles endpoint at /cgi/certifykey

```
#include "common.h"
#include "cgi.h"
#include "db.h"
#include "jwt.h"
#include "oauth.h"
#include "config.h"
#include "../cJSON/cJSON.h"
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include <string.h>
#include <errno.h>
#include <ctype.h>
```

Include dependency graph for certifykey.c:



Functions

- int main (void)

  main function for certifykey

5.2.1 Detailed Description

key signing utility. handles endpoint at /cgi/certifykey

Request: Expects a POST Request with Content-Type set to `text/plain`. The Request Header must contain a valid Authorization Bearer Token (JWT). The User must be 'active'. The Body must contain the PublicKey the User wants to get signed.

Response (if successful): Content-Type: `text/plain` The Body contains the signed PublicKey of the User.

5.2.2 Function Documentation

5.2.2.1 main()

```
int main (
            void  )
```

main function for certifykey

Returns

the exit code of this program.

Return values

0	success
!0	failure

Definition at line 174 of file certifykey.c.

References cgi_testenv(), config_s::db, ERRMSGMAXSIZE, and UUIDLEN.

Here is the call graph for this function:

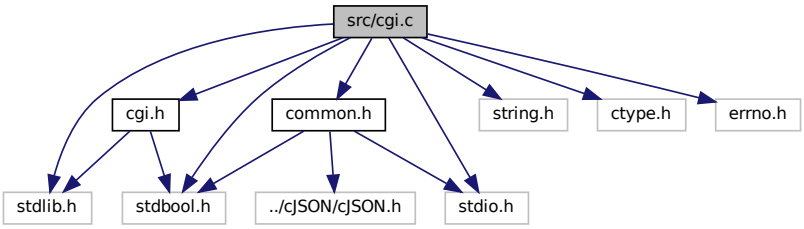
5.3 src/cgi.c File Reference

Code for working with CGI.

```
#include "cgi.h"
#include "common.h"
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include <string.h>
#include <ctype.h>
```

```
#include <errno.h>
```

Include dependency graph for cgi.c:



Functions

- struct cgi_kvp_s * cgi_parse_query_string (const char *qrystr)

  Parses a QUERY_STRING.
- struct cgi_kvp_s * cgi_parse_http_cookie (const char *cokstr)

  Parses a HTTP_COOKIE string.
- struct cgi_kvp_s * cgi_kvp_create (void)

  Create a key-value-pairs list.
- void cgi_kvp_free (struct cgi_kvp_s *kvp)

  Frees a key-value-pairs list.
- bool cgi_kvp_add (struct cgi_kvp_s *kvp, const char *key, const char *value)

  Adds a key-value-pair to a key-value-pairs list.
- bool cgi_kvp_add_delimbased (struct cgi_kvp_s *kvp, char *keyvalue, const char delim)

  Adds a key-value-pair to a key-value-pairs list but the key-value-pair is represented as a single string seperated by a delimiter.
- char * cgi_kvp_get (struct cgi_kvp_s *kvp, const char *key)

  Gets a value associated with a given key from a key-value-pairs list.
- int cgi_testenv (char *errmsg, char method, const char *post_content_type)
- void **cgi_printstatuscodeasstr** (int statuscode)
- void **cgi_printerror** (char *errmsg, int statuscode)

5.3.1 Detailed Description

Code for working with CGI.

Note

This file may get replaced in future by the REST API Code.

5.3.2 Function Documentation

5.3.2.1 cgi_kvp_add()

```
bool cgi_kvp_add (
            struct cgi_kvp_s * kvp,
            const char * key,
            const char * value )
```

Adds a key-value-pair to a key-value-pairs list.

Note

Neither key or value can be NULL.

Duplicate keys can exist and cgi_kvp_get() will only retrieve the first one. Thus you cannot override a value.

Parameters

in,out	*kvp*	the key-value-pairs list
in	*key*	the key
in	*value*	the value

Returns

whether the pair was successfully added to the list

Return values

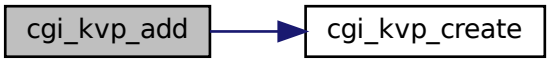
true	success
false	an error occured and errno might be set

Definition at line 179 of file cgi.c.

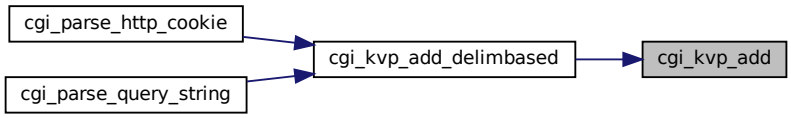
References cgi_kvp_create(), cgi_kvp_s::key, cgi_kvp_s::next, and cgi_kvp_s::value.

Referenced by cgi_kvp_add_delimbased().

Here is the call graph for this function:



Here is the caller graph for this function:



5.3.2.2 cgi_kvp_add_delimbased()

```
bool cgi_kvp_add_delimbased (
            struct cgi_kvp_s * kvp,
            char * keyvalue,
            const char delim )
```

Adds a key-value-pair to a key-value-pairs list but the key-value-pair is represented as a single string seperated by a delimiter.

If the delimiter is not or more than once in keyvalue, then 0 is returned.

This function is destructive and will replace the delimiter in keyvalue with '\0'.

The delimiter cannot be '\0'.

See also

cgi_kvp_add()

Parameters

in,out	*kvp*	the key-value-pairs list
in	*keyvalue*	the key and value in the same string seperated by the delimiter
in	*delim*	the delimiter

Returns

whether the pair was successfully added to the list

Return values

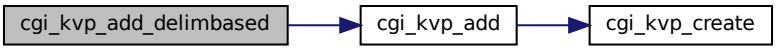
true	success
false	an error occured and errno might be set or the delimiter was not exactly once in the string

Definition at line 235 of file cgi.c.

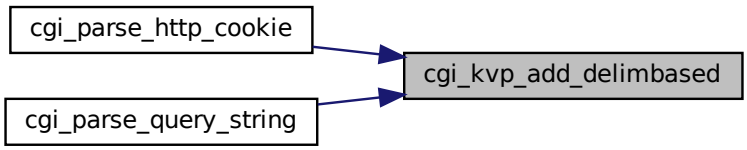
References cgi_kvp_add().

Referenced by cgi_parse_http_cookie(), and cgi_parse_query_string().

Here is the call graph for this function:



Here is the caller graph for this function:



5.3.2.3 cgi_kvp_create()

```
struct cgi_kvp_s* cgi_kvp_create (
            void  )
```

Create a key-value-pairs list.

Returns

a key-value-pairs struct

Return values

NULL	an error occurred and errno is set

The returned pointer must be freed using cgi_kvp_free().

Definition at line 131 of file cgi.c.

References cgi_kvp_s::key, cgi_kvp_s::next, and cgi_kvp_s::value.

Referenced by cgi_kvp_add(), cgi_parse_http_cookie(), and cgi_parse_query_string().

Here is the caller graph for this function:

5.3.2.4 cgi_kvp_free()

```
void cgi_kvp_free (
            struct cgi_kvp_s * kvp )
```

Frees a key-value-pairs list.

Parameters

in,out	*kvp*	the key-value-pairs list

See also

cgi_kvp_create()

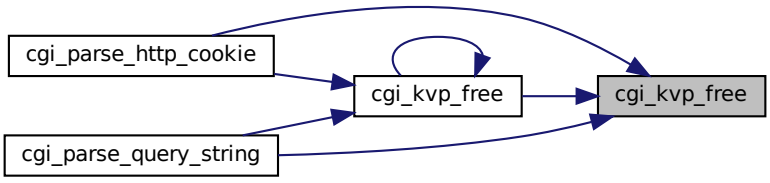
Definition at line 152 of file cgi.c.

References cgi_kvp_free(), cgi_kvp_s::key, cgi_kvp_s::next, and cgi_kvp_s::value.

Referenced by cgi_kvp_free(), cgi_parse_http_cookie(), and cgi_parse_query_string().

Here is the call graph for this function:

Here is the caller graph for this function:



5.3.2.5 cgi_kvp_get()

```
char* cgi_kvp_get (
            struct cgi_kvp_s * kvp,
            const char * key )
```

Gets a value associated with a given key from a key-value-pairs list.

Parameters

in,out	*kvp*	the key-value-pairs list
in	*key*	the key

Returns

the value associated with the key

Return values

NULL	no value is associated with the key

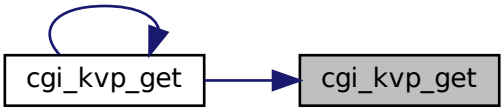
Definition at line 262 of file cgi.c.

References cgi_kvp_get(), cgi_kvp_s::key, cgi_kvp_s::next, and cgi_kvp_s::value.

Referenced by cgi_kvp_get().

Here is the call graph for this function:

Here is the caller graph for this function:



5.3.2.6 cgi_parse_http_cookie()

```
struct cgi_kvp_s* cgi_parse_http_cookie (
            const char * cokstr )
```

Parses a HTTP_COOKIE string.

This is not standard conforming!

The HTTP_COOKIE sting is a a list of key-value-pairs seperated by ';' characters. Each key-value-pair is intern seperated by an '=' character. Whitespaces are ignored.

Parameters

in	*cokstr*	the HTTP_COOKIE string.

Returns

a key-value-pairs list containing the key-value-pairs

Return values

NULL	an error occurred and errno might be set. maybe the HTTP_COOKIE string is invalid?

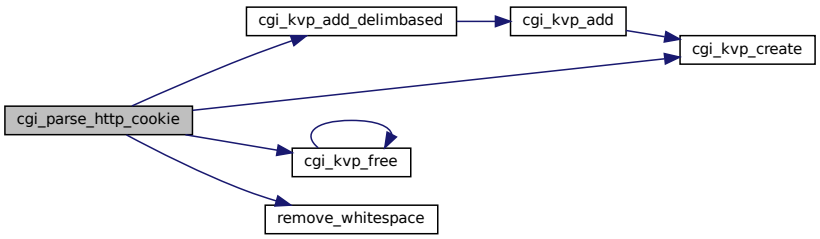
The values can be accessed via cgi_kvp_get().

The returned pointer must be freed using cgi_kvp_free().

Definition at line 86 of file cgi.c.

References cgi_kvp_add_delimbased(), cgi_kvp_create(), cgi_kvp_free(), and remove_whitespace().

Here is the call graph for this function:



5.3.2.7 cgi_parse_query_string()

```
struct cgi_kvp_s* cgi_parse_query_string (
            const char * qrystr )
```

Parses a QUERY_STRING.

This is not standard conforming!

The QUERY_STRING is a a list of key-value-pairs seperated by '&' characters. Each key-value-pair is intern seperated by an '=' character.

Parameters

in	*qrystr*	the QUERY_STRING

Returns

a key-value-pairs list containing the key-value-pairs

Return values

NULL	an error occurred and errno might be set. maybe the QUERY_STRING is invalid?

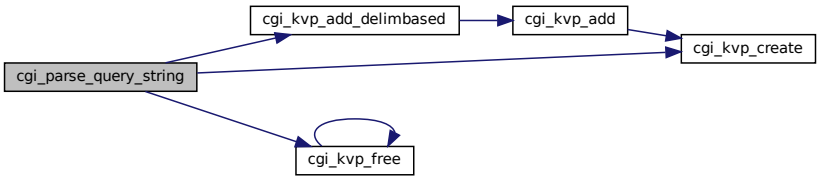
The values can be accessed via cgi_kvp_get().

The returned pointer must be freed using cgi_kvp_free().

Definition at line 35 of file cgi.c.

References cgi_kvp_add_delimbased(), cgi_kvp_create(), and cgi_kvp_free().

Here is the call graph for this function:



5.3.2.8 cgi_testenv()

```
int cgi_testenv (
            char * errmsg,
            char method,
            const char * post_content_type )
```

Parameters

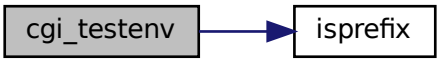
in	*method*	'p': POST 'g': GET

Definition at line 280 of file cgi.c.

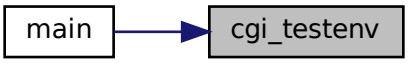
References CGI_MAXINPUTSIZE, and isprefix().

Referenced by main().

Here is the call graph for this function:



Here is the caller graph for this function:

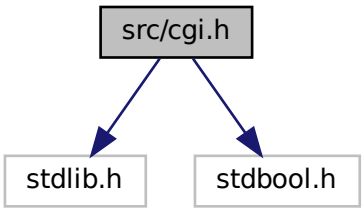


5.4 src/cgi.h File Reference

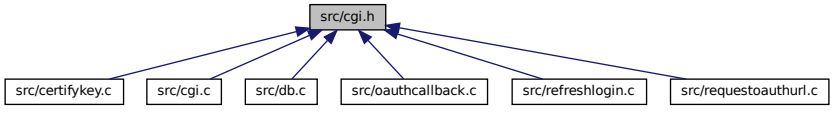
Header file for cgi.c.

```
#include <stdlib.h>
#include <stdbool.h>
```

Include dependency graph for cgi.h:



This graph shows which files directly or indirectly include this file:



Data Structures

- struct cgi_kvp_s

  key-value-pairs struct

Macros

- #define CGI_MAXINPUTSIZE 65536

  the maximum value for CONTENT_LENGTH

Functions

- struct cgi_kvp_s * cgi_parse_query_string (const char *qrystr)

  Parses a QUERY_STRING.
- struct cgi_kvp_s * cgi_parse_http_cookie (const char *cokstr)

  Parses a HTTP_COOKIE string.
- struct cgi_kvp_s * cgi_kvp_create (void)

  Create a key-value-pairs list.
- void cgi_kvp_free (struct cgi_kvp_s *kvp)

  Frees a key-value-pairs list.
- bool cgi_kvp_add (struct cgi_kvp_s *kvp, const char *key, const char *value)

  Adds a key-value-pair to a key-value-pairs list.
- bool cgi_kvp_add_delimbased (struct cgi_kvp_s *kvp, char *keyvalue, const char delim)

  Adds a key-value-pair to a key-value-pairs list but the key-value-pair is represented as a single string seperated by a delimiter.
- char * cgi_kvp_get (struct cgi_kvp_s *kvp, const char *key)

  Gets a value associated with a given key from a key-value-pairs list.
- int cgi_testenv (char *errmsg, char method, const char *post_content_type)
- void **cgi_printstatuscodeasstr** (int statuscode)
- void **cgi_printerror** (char *errmsg, int statuscode)

5.4.1 Detailed Description

Header file for cgi.c.

5.4.2 Function Documentation

5.4.2.1 cgi_kvp_add()

```
bool cgi_kvp_add (
            struct cgi_kvp_s * kvp,
            const char * key,
            const char * value )
```

Adds a key-value-pair to a key-value-pairs list.

Note

Neither key or value can be NULL.

Duplicate keys can exist and cgi_kvp_get() will only retrieve the first one. Thus you cannot override a value.

Parameters

in,out	*kvp*	the key-value-pairs list
in	*key*	the key
in	*value*	the value

Returns

whether the pair was successfully added to the list

Return values

true	success
false	an error occured and errno might be set

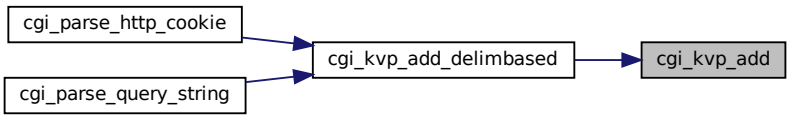
Definition at line 179 of file cgi.c.

References cgi_kvp_create(), cgi_kvp_s::key, cgi_kvp_s::next, and cgi_kvp_s::value.

Referenced by cgi_kvp_add_delimbased().

Here is the call graph for this function:

Here is the caller graph for this function:



5.4.2.2 cgi_kvp_add_delimbased()

```
bool cgi_kvp_add_delimbased (
            struct cgi_kvp_s * kvp,
            char * keyvalue,
            const char delim )
```

Adds a key-value-pair to a key-value-pairs list but the key-value-pair is represented as a single string seperated by a delimiter.

If the delimiter is not or more than once in keyvalue, then 0 is returned.

This function is destructive and will replace the delimiter in keyvalue with '\0'.

The delimiter cannot be '\0'.

See also

cgi_kvp_add()

Parameters

in,out	*kvp*	the key-value-pairs list
in	*keyvalue*	the key and value in the same string seperated by the delimiter
in	*delim*	the delimiter

Returns

whether the pair was successfully added to the list

Return values

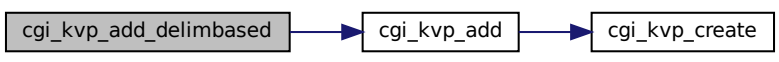
true	success
false	an error occured and errno might be set or the delimiter was not exactly once in the string

Definition at line 235 of file cgi.c.

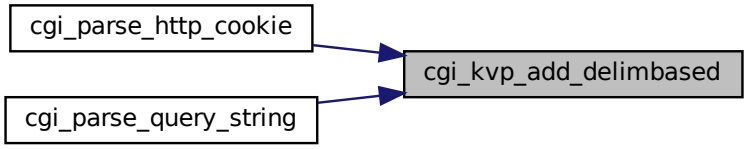
References cgi_kvp_add().

Referenced by cgi_parse_http_cookie(), and cgi_parse_query_string().

Here is the call graph for this function:



Here is the caller graph for this function:



5.4.2.3 cgi_kvp_create()

```
struct cgi_kvp_s* cgi_kvp_create (
            void  )
```

Create a key-value-pairs list.

Returns

a key-value-pairs struct

Return values

NULL	an error occurred and errno is set

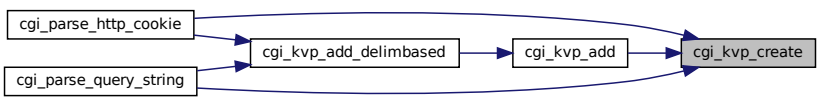
The returned pointer must be freed using cgi_kvp_free().

Definition at line 131 of file cgi.c.

References cgi_kvp_s::key, cgi_kvp_s::next, and cgi_kvp_s::value.

Referenced by cgi_kvp_add(), cgi_parse_http_cookie(), and cgi_parse_query_string().

Here is the caller graph for this function:



5.4.2.4 cgi_kvp_free()

```
void cgi_kvp_free (
            struct cgi_kvp_s * kvp )
```

Frees a key-value-pairs list.

Parameters

in,out	*kvp*	the key-value-pairs list

See also

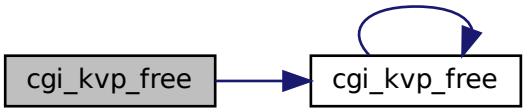
cgi_kvp_create()

Definition at line 152 of file cgi.c.

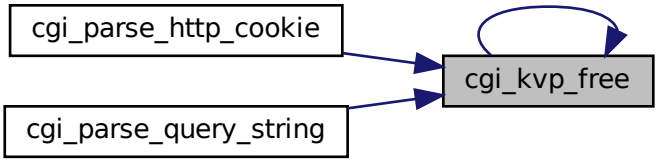
References cgi_kvp_free(), cgi_kvp_s::key, cgi_kvp_s::next, and cgi_kvp_s::value.

Referenced by cgi_kvp_free(), cgi_parse_http_cookie(), and cgi_parse_query_string().

Here is the call graph for this function:



Here is the caller graph for this function:



5.4.2.5 cgi_kvp_get()

```
char* cgi_kvp_get (
            struct cgi_kvp_s * kvp,
            const char * key )
```

Gets a value associated with a given key from a key-value-pairs list.

Parameters

`in,out`	*kvp*	the key-value-pairs list
`in`	*key*	the key

Returns

the value associated with the key

Return values

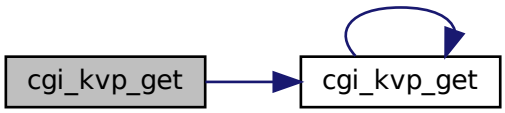
NULL	no value is associated with the key

Definition at line 262 of file cgi.c.

References cgi_kvp_get(), cgi_kvp_s::key, cgi_kvp_s::next, and cgi_kvp_s::value.

Referenced by cgi_kvp_get().

Here is the call graph for this function:



Here is the caller graph for this function:

5.4.2.6 cgi_parse_http_cookie()

```
struct cgi_kvp_s* cgi_parse_http_cookie (
            const char * cokstr )
```

Parses a HTTP_COOKIE string.

This is not standard conforming!

The HTTP_COOKIE sting is a a list of key-value-pairs seperated by ';' characters. Each key-value-pair is intern seperated by an '=' character. Whitespaces are ignored.

Parameters

in	*cokstr*	the HTTP_COOKIE string.

Returns

a key-value-pairs list containing the key-value-pairs

Return values

NULL	an error occurred and errno might be set. maybe the HTTP_COOKIE string is invalid?

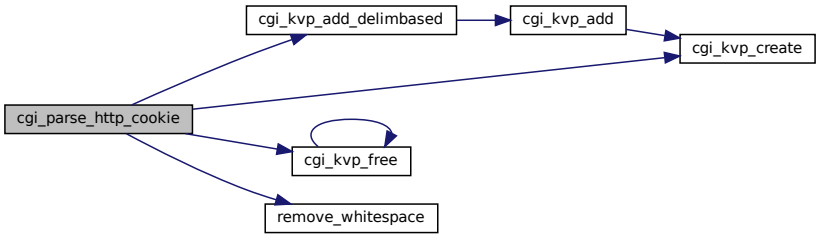
The values can be accessed via cgi_kvp_get().

The returned pointer must be freed using cgi_kvp_free().

Definition at line 86 of file cgi.c.

References cgi_kvp_add_delimbased(), cgi_kvp_create(), cgi_kvp_free(), and remove_whitespace().

Here is the call graph for this function:



5.4.2.7 cgi_parse_query_string()

```
struct cgi_kvp_s* cgi_parse_query_string (
            const char * qrystr )
```

Parses a QUERY_STRING.

This is not standard conforming!

The QUERY_STRING is a a list of key-value-pairs seperated by '&' characters. Each key-value-pair is intern seperated by an '=' character.

Parameters

in	*qrystr*	the QUERY_STRING

Returns

a key-value-pairs list containing the key-value-pairs

Return values

NULL	an error occurred and errno might be set. maybe the QUERY_STRING is invalid?

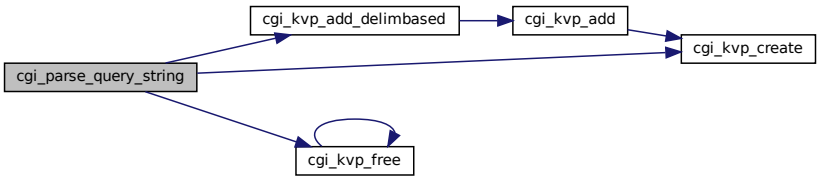
The values can be accessed via cgi_kvp_get().

The returned pointer must be freed using cgi_kvp_free().

Definition at line 35 of file cgi.c.

References cgi_kvp_add_delimbased(), cgi_kvp_create(), and cgi_kvp_free().

Here is the call graph for this function:



5.4.2.8 cgi_testenv()

```
int cgi_testenv (
            char * errmsg,
            char method,
            const char * post_content_type )
```

Parameters

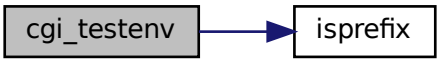
in	*method*	'p': POST 'g': GET

Definition at line 280 of file cgi.c.

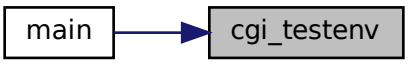
References CGI_MAXINPUTSIZE, and isprefix().

Referenced by main().

Here is the call graph for this function:



Here is the caller graph for this function:

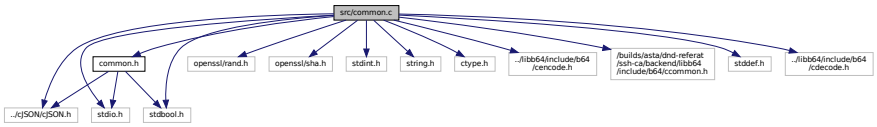


5.5 src/common.c File Reference

A bunch of useful/common code.

```
#include "common.h"
#include <openssl/rand.h>
#include <openssl/sha.h>
#include <stdio.h>
#include <stdint.h>
#include <stdbool.h>
#include <string.h>
#include <ctype.h>
#include "../cJSON/cJSON.h"
#include "../libb64/include/b64/cencode.h"
#include "../libb64/include/b64/cdecode.h"
```

Include dependency graph for common.c:



Functions

- bool uuid_v4_gen (char *buffer)

  Generate a Version 4 UUID according to RFC-4122.
- bool is_valid_uuid_v4 (const char *buffer)

  Tests whether a given string is a valid UUIDv4.
- bool is_valid_hex (char c)

  Tests whether a character is a valid hexadecimal character.
- int hex_to_int (char c)

  Converts a hexadecimal character to a number.
- char int_to_hex (int x)

  Converts a number to an uppercase hexadecimal character.
- bool is_percent_encoded (const char *str)

  Tests whether a given string is correctly percent encoded.
- char * percent_decode (const char *str)

  Decodes a percent encoded string.
- char * percent_encode (const char *str)

  Percent encodes a string.
- bool isprefix (const char *pre, const char *str)

  Tests whether str starts with pre.
- char * filetostr (const char *path)

  Reads a file into a string.
- cJSON * filetojson (const char *path)

  Reads a JSON file.
- char * base64_encode (const char *str, size_t len)

  base64 encodes a string
- char * base64_decode (const char *str)

  decodes a base64 encoded string
- char * base64url_encode (const char *str, size_t len, bool skippadding)

  base64url encodes a string
- char * base64url_decode (char *str)

  decodes a base64url encoded string
- bool hmac_sha256 (unsigned char *hash, const char *message, const char *key)

  Calculates the HMAC-SHA256 of a given message with a given key.
- void remove_whitespace (char *s)

  Removes all whitespace characters from a string.
- bool writestreamtofile (FILE *stream, const char *outpath)

  Writes a stream to the file located at outpath.
- bool get_string_from_json (char *errmsg, const cJSON *json, const char *curjsonkey, const char *stringkey, char **res, bool can_be_null)

  Tries to retrieve the value of stringkey from json as a string.
- bool get_number_from_json (char *errmsg, const cJSON *json, const char *curjsonkey, const char *numberkey, long *res)

  Tries to retrieve the value of numberkey from json as a long.
- bool get_bool_from_json (char *errmsg, const cJSON *json, const char *curjsonkey, const char *boolkey, bool *res)

  Tries to retrieve the value of boolkey from json as an int.
- char * str_append (char *dest, const char *src, size_t *destlen, size_t srclen)

  Resizes the dest string and appends the src string to the dest string.

5.5.1 Detailed Description

A bunch of useful/common code.

5.5.2 Function Documentation

5.5.2.1 base64_decode()

```
char* base64_decode (
            const char * str )
```

decodes a base64 encoded string

Uses libb64 to decode a string.

Parameters

in	*str*	the encoded input string to be decoded

Returns

the decoded string

Return values

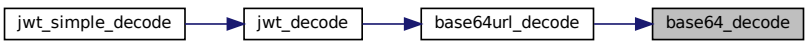
NULL	an error occurred and errno might be set

The returned string must be freed using free().

Definition at line 477 of file common.c.

Referenced by base64url_decode().

Here is the caller graph for this function:



5.5.2.2 base64_encode()

```
char* base64_encode (
            const char * str,
            size_t len )
```

base64 encodes a string

Uses libb64 to encode a string.

Parameters

in	*str*	the input string to be encoded
in	*len*	the length of the string (if str isn't raw binary this should be strlen(str))

Returns

the encoded string

Return values

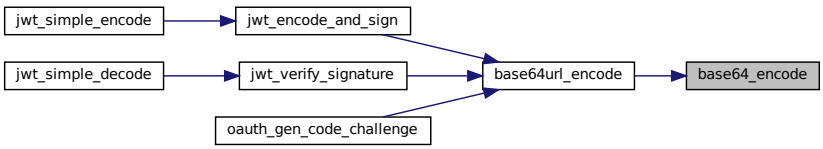
NULL	an error occurred and errno might be set

The returned string must be freed using free().

Definition at line 419 of file common.c.

Referenced by base64url_encode().

Here is the caller graph for this function:



5.5.2.3 base64url_decode()

```
char* base64url_decode (
            char * str )
```

decodes a base64url encoded string

Conforming to RFC 4648 §5.

This function will change str! This is because you will probably use this function with the result of a percent_decode() call.

Note

This function expects the padding to be using '=' characters.

Parameters

in,out	*str*	the encoded input string to be decoded

Returns

the decoded string

Return values

NULL	an error occurred and errno might be set

The returned string must be freed using free().

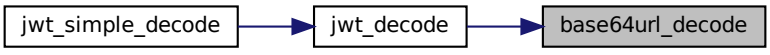
Definition at line 574 of file common.c.

References base64_decode().

Referenced by jwt_decode().

Here is the call graph for this function:

Here is the caller graph for this function:



5.5.2.4 base64url_encode()

```
char* base64url_encode (
            const char * str,
            size_t len,
            bool skippadding )
```

base64url encodes a string

Conforming to RFC 4648 §5.

Note

You might wish to use percent_encode() on the result of this function before passing it over into a url, as padding is still using '=' characters. Or you just disable padding with skippadding.

Parameters

in	*str*	the input string to be encoded
in	*len*	the length of the string (if str isn't raw binary this should be strlen(str))
in	*skippadding*	whether the padding at the end should be skipped

Returns

the encoded string

Return values

NULL	an error occurred and errno might be set

The returned string must be freed using free().

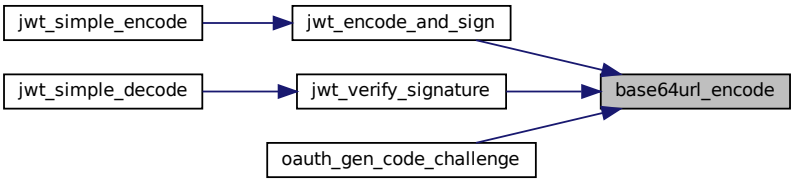
Definition at line 538 of file common.c.

References base64_encode().

Referenced by jwt_encode_and_sign(), jwt_verify_signature(), and oauth_gen_code_challenge().

Here is the call graph for this function:

Here is the caller graph for this function:



5.5.2.5 filetojson()

```
cJSON* filetojson (
            const char * path )
```

Reads a JSON file.

Reads the file at path and parses it into a cJSON object.

Returns

The files contents as a cJSON object.

Return values

NULL	an error occurred and errno might be set.

The returned cJSON object must be freed using cJSON_Delete().

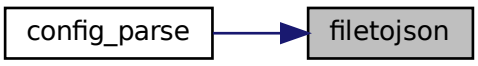
Definition at line 395 of file common.c.

References filetostr().

Referenced by config_parse().

Here is the call graph for this function:

Here is the caller graph for this function:



5.5.2.6 filetostr()

```
char* filetostr (
            const char * path )
```

Reads a file into a string.

Reads the file at path and converts it into a null-terminated string.

Note

This function is not suitable for binary files that may contain '\0' characters.

Returns

The files contents as a string.

Return values

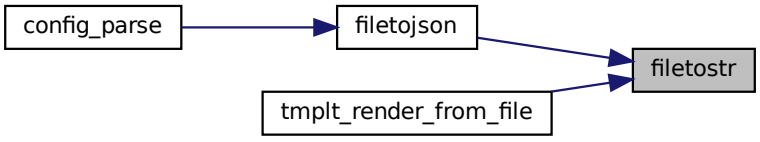
NULL	an error occurred and errno might be set.

The returned string must be freed using free().

Definition at line 358 of file common.c.

Referenced by filetojson(), and tmplt_render_from_file().

Here is the caller graph for this function:



5.5.2.7 get_bool_from_json()

```
bool get_bool_from_json (
            char * errmsg,
            const cJSON * json,
            const char * curjsonkey,
            const char * boolkey,
            bool * res )
```

Tries to retrieve the value of boolkey from json as an int.

Parameters

out	*errmsg*	buffer for error messages. size: ERRMSGMAXSIZE
in	*json*	the json object that is supposed to contain boolkey
in	*curjsonkey*	the name of json (used for error messages)
in	*boolkey*	the name of the key
out	*res*	where the retrieved value should be stored

Returns

whether the function was successful.

Return values

true	success. a pointer to the value has been stored in res
false	failed retrieving the bool and an error message has been written to errmsg

Definition at line 839 of file common.c.

5.5.2.8 get_number_from_json()

```
bool get_number_from_json (
            char * errmsg,
            const cJSON * json,
            const char * curjsonkey,
            const char * numberkey,
            long * res )
```

Tries to retrieve the value of numberkey from json as a long.

Parameters

out	*errmsg*	buffer for error messages. size: ERRMSGMAXSIZE
in	*json*	the json object that is supposed to contain numberkey
in	*curjsonkey*	the name of json (used for error messages)
in	*numberkey*	the name of the key
out	*res*	where the retrieved value should be stored

Returns

whether the function was successful.

Return values

true	success. a pointer to the value has been stored in res
false	failed retrieving the number and an error message has been written to errmsg

Definition at line 802 of file common.c.

5.5.2.9 get_string_from_json()

```
bool get_string_from_json (
            char * errmsg,
            const cJSON * json,
            const char * curjsonkey,
            const char * stringkey,
            char ** res,
            bool can_be_null )
```

Tries to retrieve the value of stringkey from json as a string.

Note: res will only contain the pointer to the string inside the json object and thus will be freed by calling cJSON_↩ Delete on json.

Parameters

out	*errmsg*	buffer for error messages. size: ERRMSGMAXSIZE
in	*json*	the json object that is supposed to contain stringkey
in	*curjsonkey*	the name of json (used for error messages)
in	*stringkey*	the name of the key
out	*res*	pointer to the pointer where the pointer of the retrieved value should be stored
in	*can_be_null*	whether the value is allowed to be null

Returns

whether the function was successful.

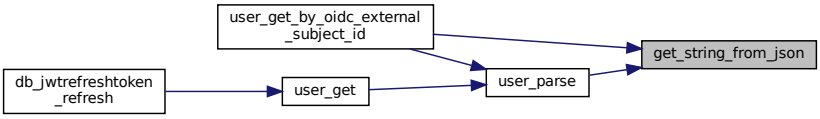
Return values

true	success. a pointer to the value has been stored in res
false	failed retrieving the string and an error message has been written to errmsg

Definition at line 758 of file common.c.

Referenced by user_get_by_oidc_external_subject_id(), and user_parse().

Here is the caller graph for this function:



5.5.2.10 hex_to_int()

```
int hex_to_int (
            char c )
```

Converts a hexadecimal character to a number.

Basically '0'-'9' are mapped to 0-9 and 'a'-'f' or 'A'-'F' are mapped to 10-15.

A non-hexadecimal character is just treated as '0'.

Parameters

in	*c*	the character

Returns

c as a number

Definition at line 150 of file common.c.

5.5.2.11 hmac_sha256()

```
bool hmac_sha256 (
            unsigned char * hash,
            const char * message,
            const char * key )
```

Calculates the HMAC-SHA256 of a given message with a given key.

See also

https://en.wikipedia.org/wiki/HMAC

Parameters

out	*hash*	buffer for the resulting hash; Must be at least SHA256_DIGEST_LENGTH bytes long.
in	*message*	the message
in	*key*	the key

Returns

whether the function was successful.

Return values

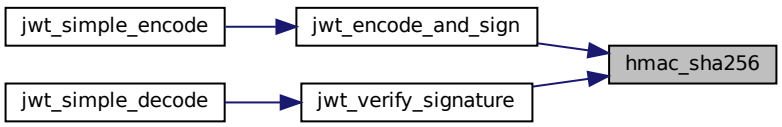
true	success
false	an error occurred and errno might be set

Definition at line 609 of file common.c.

References cgi_kvp_s::key.

Referenced by jwt_encode_and_sign(), and jwt_verify_signature().

Here is the caller graph for this function:



5.5.2.12 int_to_hex()

```
char int_to_hex (
            int x )
```

Converts a number to an uppercase hexadecimal character.

Basically 0-9 are mapped to '0'-'9' and 10-15 are mapped to 'A'-'F'.

A number larger than 15 or smaller than 0 is just treated as 0.

Parameters

in	*x*	the number

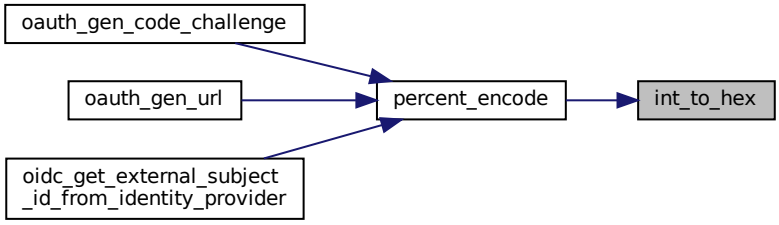
Returns

x as an uppercase hexadecimal character

Definition at line 173 of file common.c.

Referenced by percent_encode().

Here is the caller graph for this function:



5.5.2.13 is_percent_encoded()

```
bool is_percent_encoded (
            const char * str )
```

Tests whether a given string is correctly percent encoded.

Conforming to RFC 3986 §2.1.

Parameters

in	*str*	the string to be tested

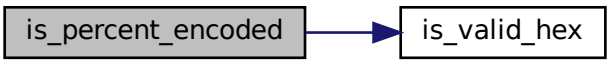
Returns

whether str is correctly percent encoded.

Definition at line 190 of file common.c.

References is_valid_hex().

Here is the call graph for this function:



5.5.2.14 is_valid_hex()

```
bool is_valid_hex (
            char c )
```

Tests whether a character is a valid hexadecimal character.

Valid hexadecimal characters are: 0-9 a-f A-F

Parameters

in	*c*	the character to be tested

Returns

whether c is a valid hex character.

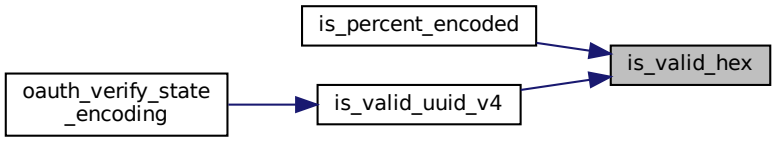
Return values

false	not a valid UUIDv4
true	a valid UUIDv4

Definition at line 132 of file common.c.

Referenced by is_percent_encoded(), and is_valid_uuid_v4().

Here is the caller graph for this function:



5.5.2.15 is_valid_uuid_v4()

```
bool is_valid_uuid_v4 (
            const char * buffer )
```

Tests whether a given string is a valid UUIDv4.

A UUIDv4 looks like this: 034d9d19-0182-4fd8-aa18-21ffd18bf956

See also

uuid_v4_gen()

Parameters

in	*buffer*	the string to be tested

Returns

whether buffer is a valid UUIDv4.

Return values

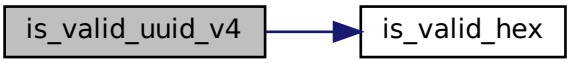
false	not a valid UUIDv4
true	a valid UUIDv4

Definition at line 82 of file common.c.

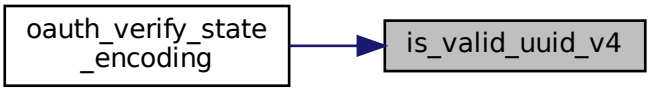
References is_valid_hex(), and UUIDLEN.

Referenced by oauth_verify_state_encoding().

Here is the call graph for this function:



Here is the caller graph for this function:



5.5.2.16 isprefix()

```
bool isprefix (
            const char * pre,
            const char * str )
```

Tests whether str starts with pre.

Parameters

in	*pre*	the expected prefix string
in	*str*	the string to be tested

Returns

whether str starts with pre.

Definition at line 341 of file common.c.

Referenced by cgi_testenv().

Here is the caller graph for this function:

5.5.2.17 percent_decode()

```
char* percent_decode (
            const char * str )
```

Decodes a percent encoded string.

This function expects str to be conforming to RFC 3986 §2.1. You can test if str is valid using is_percent_encoded().

Note

The sequence "%00" may cause undefined behaviour as it will just null-terminate the string at that point.

Parameters

in	*str*	the percent encoded string to be decoded

Returns

the decoded string

Return values

NULL	an error occurred and errno might be set

The returned string must be freed using free().

Definition at line 251 of file common.c.

5.5.2.18 percent_encode()

```
char* percent_encode (
            const char * str )
```

Percent encodes a string.

Conforming to RFC 3986 §2.1.

Note

This function cannot handle '\0' characters as those are used for indicating the end of a string in C.

Parameters

in	*str*	the string to be encoded

Returns

the encoded string

Return values

NULL	an error occurred and errno might be set

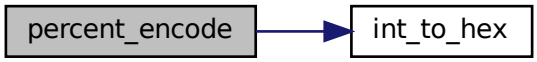
The returned string must be freed using free().

Definition at line 296 of file common.c.

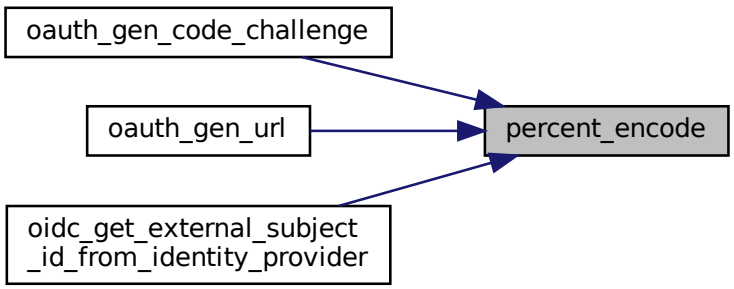
References int_to_hex().

Referenced by oauth_gen_code_challenge(), oauth_gen_url(), and oidc_get_external_subject_id_from_identity_↩provider().

Here is the call graph for this function:



Here is the caller graph for this function:



5.5.2.19 remove_whitespace()

```
void remove_whitespace (
            char * s )
```

Removes all whitespace characters from a string.

Parameters

in,out	*s*	the string

Definition at line 680 of file common.c.

Referenced by cgi_parse_http_cookie().

Here is the caller graph for this function:

5.5.2.20 str_append()

```
char* str_append (
            char * dest,
            const char * src,
            size_t * destlen,
            size_t srclen )
```

Resizes the dest string and appends the src string to the dest string.

Uses realloc() to resize dest to fit dest, src and the terminating null byte, then it appends srclen characters of the src string to the dest string, overwriting the terminating null byte at the end of dest, and then adds a terminating null byte.

Note

dest and src must not be NULL.

Parameters

in,out	*dest*	the destination string
in	*src*	the source string
in,out	*destlen*	pointer where the current length of dest is stored. This function also updates this value. Excluding the terminating null byte. Ignored if NULL.
in	*srclen*	the length of src. This function will use strlen(src), if srclen == 0.

Returns

the new location of dest.

Return values

NULL	an error occurred and errno might be set.

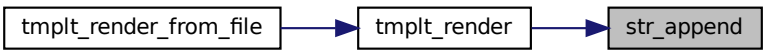
Note

dest is freed by this function if an error occurres.

Definition at line 884 of file common.c.

Referenced by tmplt_render().

Here is the caller graph for this function:



5.5.2.21 uuid_v4_gen()

```
bool uuid_v4_gen (
             char * buffer )
```

Generate a Version 4 UUID according to RFC-4122.

Uses the openssl RAND_bytes function to generate a Version 4 UUID.

Parameters

in,out	*buffer*	A buffer that is at least (UUIDLEN + 1) bytes long.

Return values

true	success
false	failure

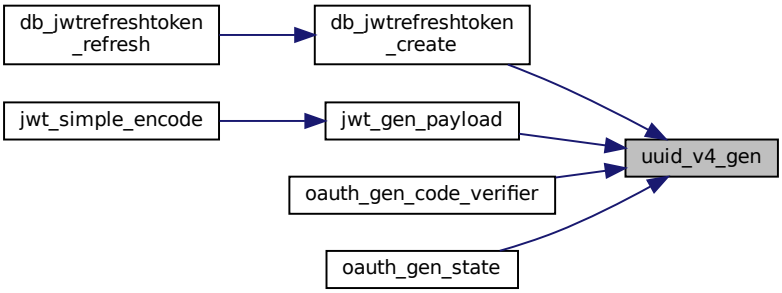
Note

Code from https://gist.github.com/kvelakur/9069c9896577c3040030

Definition at line 35 of file common.c.

Referenced by db_jwtrefreshtoken_create(), jwt_gen_payload(), oauth_gen_code_verifier(), and oauth_gen_↩ state().

Here is the caller graph for this function:



5.5.2.22 writestreamtofile()

```
bool writestreamtofile (
            FILE * stream,
            const char * outpath )
```

Writes a stream to the file located at outpath.

The stream is read until EOF is reached.

Parameters

in	*stream*	the stream to be read from
in	*outpath*	path to the file were the output should be written to

Returns

whether the function was successful

Return values

true	success
false	An error occurred and errno might be set

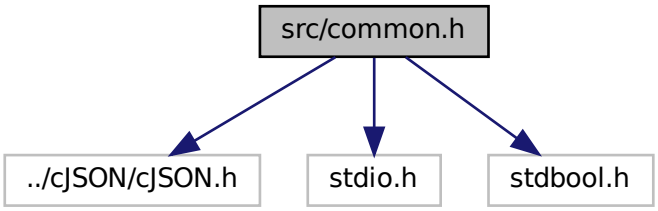
Definition at line 711 of file common.c.

5.6 src/common.h File Reference

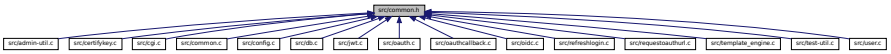
Header file for common.c.

```
#include "../cJSON/cJSON.h"
#include <stdio.h>
#include <stdbool.h>
```

Include dependency graph for common.h:



This graph shows which files directly or indirectly include this file:



Macros

- #define COLOR_BLACK "\033[0;30m"

  ANSI Escape Sequence for the color black.
- #define COLOR_RED "\033[0;31m"

  ANSI Escape Sequence for the color red.
- #define COLOR_GREEN "\033[0;32m"

  ANSI Escape Sequence for the color green.
- #define COLOR_YELLOW "\033[0;33m"

  ANSI Escape Sequence for the color yellow.
- #define COLOR_BLUE "\033[0;34m"

  ANSI Escape Sequence for the color blue.
- #define COLOR_PURPLE "\033[0;35m"

  ANSI Escape Sequence for the color purple.
- #define COLOR_CYAN "\033[0;36m"

  ANSI Escape Sequence for the color cyan.
- #define COLOR_WHITE "\033[0;37m"

  ANSI Escape Sequence for the color white.
- #define COLOR_RESET "\033[0m"

  ANSI Escape Sequence for resetting the color.
- #define CONFIGFILEPATH "/opt/ssh-ca/backend/config.json"

  path to the config file
- #define ERRMSGMAXSIZE 1024

  size of errmsg buffer
- #define UUIDLEN 36

  the length of a UUID
- #define UNUSED(x) (void)(x)

  macro for marking a variable as unused

Functions

- bool uuid_v4_gen (char *buffer)

  Generate a Version 4 UUID according to RFC-4122.
- bool is_valid_uuid_v4 (const char *buffer)

  Tests whether a given string is a valid UUIDv4.
- bool is_valid_hex (char c)

  Tests whether a character is a valid hexadecimal character.
- int hex_to_int (char c)

  Converts a hexadecimal character to a number.
- char int_to_hex (int x)

  Converts a number to an uppercase hexadecimal character.
- bool is_percent_encoded (const char *str)

  Tests whether a given string is correctly percent encoded.
- char * percent_decode (const char *str)

  Decodes a percent encoded string.
- char * percent_encode (const char *str)

  Percent encodes a string.
- bool isprefix (const char *pre, const char *str)

  Tests whether str starts with pre.
- char * filetostr (const char *path)

  Reads a file into a string.
- cJSON * filetojson (const char *path)

  Reads a JSON file.
- char * base64_encode (const char *str, size_t len)

  base64 encodes a string
- char * base64_decode (const char *str)

  decodes a base64 encoded string
- char * base64url_encode (const char *str, size_t len, bool skippadding)

  base64url encodes a string
- char * base64url_decode (char *str)

  decodes a base64url encoded string
- bool hmac_sha256 (unsigned char *hash, const char *message, const char *key)

  Calculates the HMAC-SHA256 of a given message with a given key.
- void remove_whitespace (char *s)

  Removes all whitespace characters from a string.
- bool writestreamtofile (FILE *stream, const char *outpath)

  Writes a stream to the file located at outpath.
- bool get_string_from_json (char *errmsg, const cJSON *json, const char *curjsonkey, const char *stringkey, char **res, bool can_be_null)

  Tries to retrieve the value of stringkey from json as a string.
- bool get_number_from_json (char *errmsg, const cJSON *json, const char *curjsonkey, const char *numberkey, long *res)

  Tries to retrieve the value of numberkey from json as a long.
- bool get_bool_from_json (char *errmsg, const cJSON *json, const char *curjsonkey, const char *boolkey, bool *res)

  Tries to retrieve the value of boolkey from json as an int.
- char * str_append (char *dest, const char *src, size_t *destlen, size_t srclen)

  Resizes the dest string and appends the src string to the dest string.

5.6.1 Detailed Description

Header file for common.c.

5.6.2 Function Documentation

5.6.2.1 base64_decode()

```
char* base64_decode (
            const char * str )
```

decodes a base64 encoded string

Uses libb64 to decode a string.

Parameters

in	*str*	the encoded input string to be decoded

Returns

the decoded string

Return values

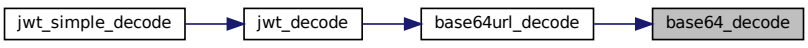
NULL	an error occurred and errno might be set

The returned string must be freed using free().

Definition at line 477 of file common.c.

Referenced by base64url_decode().

Here is the caller graph for this function:



5.6.2.2 base64_encode()

```
char* base64_encode (
            const char * str,
            size_t len )
```

base64 encodes a string

Uses libb64 to encode a string.

Parameters

in	*str*	the input string to be encoded
in	*len*	the length of the string (if str isn't raw binary this should be strlen(str))

Returns

the encoded string

Return values

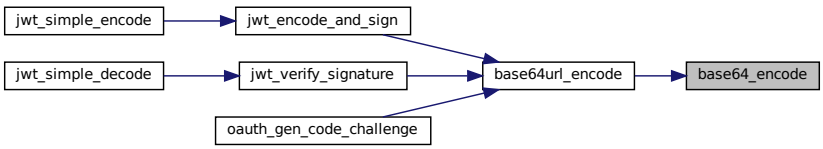
NULL	an error occurred and errno might be set

The returned string must be freed using free().

Definition at line 419 of file common.c.

Referenced by base64url_encode().

Here is the caller graph for this function:



5.6.2.3 base64url_decode()

```
char* base64url_decode (
            char * str )
```

decodes a base64url encoded string

Conforming to RFC 4648 §5.

This function will change str! This is because you will probably use this function with the result of a percent_decode() call.

Note

This function expects the padding to be using '=' characters.

Parameters

in,out	*str*	the encoded input string to be decoded

Returns

the decoded string

Return values

NULL	an error occurred and errno might be set

The returned string must be freed using free().

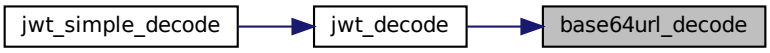
Definition at line 574 of file common.c.

References base64_decode().

Referenced by jwt_decode().

Here is the call graph for this function:

Here is the caller graph for this function:



5.6.2.4 base64url_encode()

```
char* base64url_encode (
            const char * str,
            size_t len,
            bool skippadding )
```

base64url encodes a string

Conforming to RFC 4648 §5.

Note

You might wish to use percent_encode() on the result of this function before passing it over into a url, as padding is still using '=' characters. Or you just disable padding with skippadding.

Parameters

in	*str*	the input string to be encoded
in	*len*	the length of the string (if str isn't raw binary this should be strlen(str))
in	*skippadding*	whether the padding at the end should be skipped

Returns

the encoded string

Return values

NULL	an error occurred and errno might be set

The returned string must be freed using free().

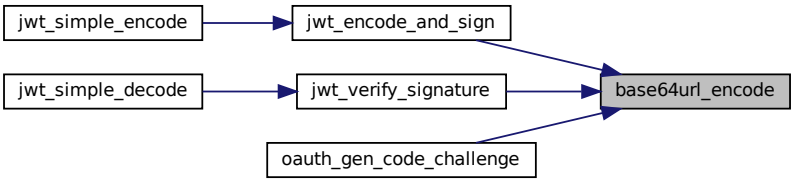
Definition at line 538 of file common.c.

References base64_encode().

Referenced by jwt_encode_and_sign(), jwt_verify_signature(), and oauth_gen_code_challenge().

Here is the call graph for this function:

Here is the caller graph for this function:



5.6.2.5 filetojson()

```
cJSON* filetojson (
            const char * path )
```

Reads a JSON file.

Reads the file at path and parses it into a cJSON object.

Returns

The files contents as a cJSON object.

Return values

NULL	an error occurred and errno might be set.

The returned cJSON object must be freed using cJSON_Delete().

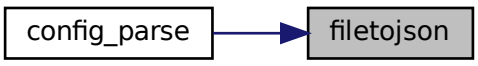
Definition at line 395 of file common.c.

References filetostr().

Referenced by config_parse().

Here is the call graph for this function:

Here is the caller graph for this function:



5.6.2.6 filetostr()

```
char* filetostr (
            const char * path )
```

Reads a file into a string.

Reads the file at path and converts it into a null-terminated string.

Note

This function is not suitable for binary files that may contain '\0' characters.

Returns

The files contents as a string.

Return values

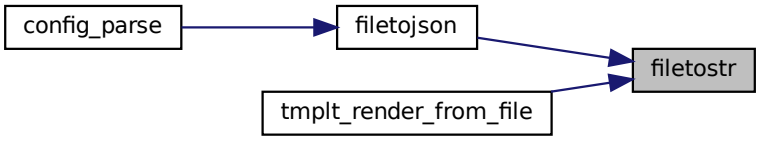
NULL	an error occurred and errno might be set.

The returned string must be freed using free().

Definition at line 358 of file common.c.

Referenced by filetojson(), and tmplt_render_from_file().

Here is the caller graph for this function:



5.6.2.7 get_bool_from_json()

```
bool get_bool_from_json (
            char * errmsg,
            const cJSON * json,
            const char * curjsonkey,
            const char * boolkey,
            bool * res )
```

Tries to retrieve the value of boolkey from json as an int.

Parameters

out	*errmsg*	buffer for error messages. size: ERRMSGMAXSIZE
in	*json*	the json object that is supposed to contain boolkey
in	*curjsonkey*	the name of json (used for error messages)
in	*boolkey*	the name of the key
out	*res*	where the retrieved value should be stored

Returns

whether the function was successful.

Return values

true	success. a pointer to the value has been stored in res
false	failed retrieving the bool and an error message has been written to errmsg

Definition at line 839 of file common.c.

5.6.2.8 get_number_from_json()

```
bool get_number_from_json (
            char * errmsg,
            const cJSON * json,
            const char * curjsonkey,
            const char * numberkey,
            long * res )
```

Tries to retrieve the value of numberkey from json as a long.

Parameters

out	*errmsg*	buffer for error messages. size: ERRMSGMAXSIZE
in	*json*	the json object that is supposed to contain numberkey
in	*curjsonkey*	the name of json (used for error messages)
in	*numberkey*	the name of the key
out	*res*	where the retrieved value should be stored

Returns

whether the function was successful.

Return values

true	success. a pointer to the value has been stored in res
false	failed retrieving the number and an error message has been written to errmsg

Definition at line 802 of file common.c.

5.6.2.9 get_string_from_json()

```
bool get_string_from_json (
            char * errmsg,
            const cJSON * json,
            const char * curjsonkey,
            const char * stringkey,
            char ** res,
            bool can_be_null )
```

Tries to retrieve the value of stringkey from json as a string.

Note: res will only contain the pointer to the string inside the json object and thus will be freed by calling cJSON_↩ Delete on json.

Parameters

out	*errmsg*	buffer for error messages. size: ERRMSGMAXSIZE
in	*json*	the json object that is supposed to contain stringkey
in	*curjsonkey*	the name of json (used for error messages)
in	*stringkey*	the name of the key
out	*res*	pointer to the pointer where the pointer of the retrieved value should be stored
in	*can_be_null*	whether the value is allowed to be null

Returns

whether the function was successful.

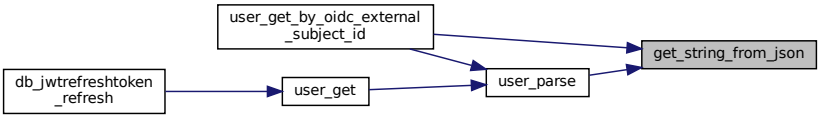
Return values

true	success. a pointer to the value has been stored in res
false	failed retrieving the string and an error message has been written to errmsg

Definition at line 758 of file common.c.

Referenced by user_get_by_oidc_external_subject_id(), and user_parse().

Here is the caller graph for this function:



5.6.2.10 hex_to_int()

```
int hex_to_int (
            char c )
```

Converts a hexadecimal character to a number.

Basically '0'-'9' are mapped to 0-9 and 'a'-'f' or 'A'-'F' are mapped to 10-15.

A non-hexadecimal character is just treated as '0'.

Parameters

in	*c*	the character

Returns

c as a number

Definition at line 150 of file common.c.

5.6.2.11 hmac_sha256()

```
bool hmac_sha256 (
            unsigned char * hash,
            const char * message,
            const char * key )
```

Calculates the HMAC-SHA256 of a given message with a given key.

See also

https://en.wikipedia.org/wiki/HMAC

Parameters

out	*hash*	buffer for the resulting hash; Must be at least SHA256_DIGEST_LENGTH bytes long.
in	*message*	the message
in	*key*	the key

Returns

whether the function was successful.

Return values

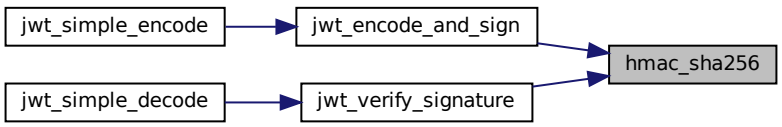
true	success
false	an error occurred and errno might be set

Definition at line 609 of file common.c.

References cgi_kvp_s::key.

Referenced by jwt_encode_and_sign(), and jwt_verify_signature().

Here is the caller graph for this function:



5.6.2.12 int_to_hex()

```
char int_to_hex (
             int x )
```

Converts a number to an uppercase hexadecimal character.

Basically 0-9 are mapped to '0'-'9' and 10-15 are mapped to 'A'-'F'.

A number larger than 15 or smaller than 0 is just treated as 0.

Parameters

in	*x*	the number

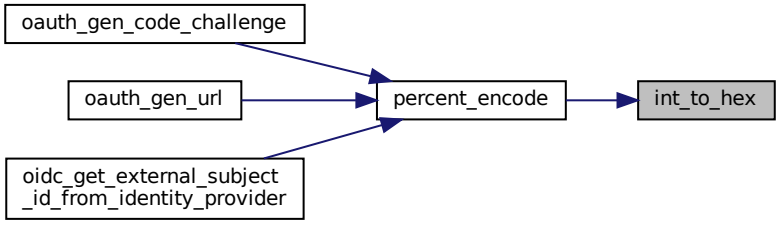
Returns

x as an uppercase hexadecimal character

Definition at line 173 of file common.c.

Referenced by percent_encode().

Here is the caller graph for this function:



5.6.2.13 is_percent_encoded()

```
bool is_percent_encoded (
            const char * str )
```

Tests whether a given string is correctly percent encoded.

Conforming to RFC 3986 §2.1.

Parameters

in	*str*	the string to be tested

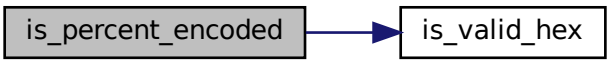
Returns

whether str is correctly percent encoded.

Definition at line 190 of file common.c.

References is_valid_hex().

Here is the call graph for this function:



5.6.2.14 is_valid_hex()

```
bool is_valid_hex (
            char c )
```

Tests whether a character is a valid hexadecimal character.

Valid hexadecimal characters are: 0-9 a-f A-F

Parameters

in	*c*	the character to be tested

Returns

whether c is a valid hex character.

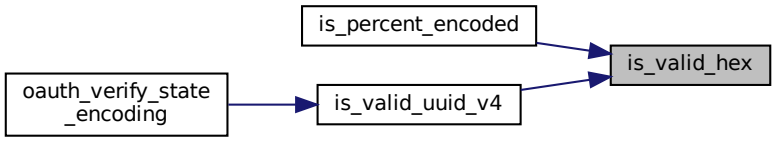
Return values

false	not a valid UUIDv4
true	a valid UUIDv4

Definition at line 132 of file common.c.

Referenced by is_percent_encoded(), and is_valid_uuid_v4().

Here is the caller graph for this function:



5.6.2.15 is_valid_uuid_v4()

```
bool is_valid_uuid_v4 (
            const char * buffer )
```

Tests whether a given string is a valid UUIDv4.

A UUIDv4 looks like this: 034d9d19-0182-4fd8-aa18-21ffd18bf956

See also

uuid_v4_gen()

Parameters

in	*buffer*	the string to be tested

Returns

whether buffer is a valid UUIDv4.

Return values

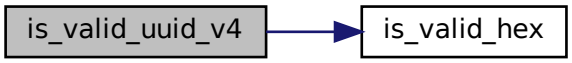
false	not a valid UUIDv4
true	a valid UUIDv4

Definition at line 82 of file common.c.

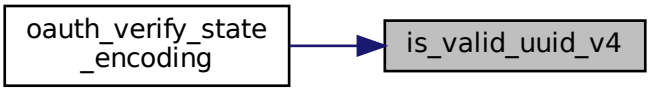
References is_valid_hex(), and UUIDLEN.

Referenced by oauth_verify_state_encoding().

Here is the call graph for this function:



Here is the caller graph for this function:



5.6.2.16 isprefix()

```
bool isprefix (
            const char * pre,
            const char * str )
```

Tests whether str starts with pre.

Parameters

in	*pre*	the expected prefix string
in	*str*	the string to be tested

Returns

whether str starts with pre.

Definition at line 341 of file common.c.

Referenced by cgi_testenv().

Here is the caller graph for this function:

5.6.2.17 percent_decode()

```
char* percent_decode (
            const char * str )
```

Decodes a percent encoded string.

This function expects str to be conforming to RFC 3986 §2.1. You can test if str is valid using is_percent_encoded().

Note

The sequence "%00" may cause undefined behaviour as it will just null-terminate the string at that point.

Parameters

in	*str*	the percent encoded string to be decoded

Returns

the decoded string

Return values

NULL	an error occurred and errno might be set

The returned string must be freed using free().

Definition at line 251 of file common.c.

5.6.2.18 percent_encode()

```
char* percent_encode (
            const char * str )
```

Percent encodes a string.

Conforming to RFC 3986 §2.1.

Note

This function cannot handle '\0' characters as those are used for indicating the end of a string in C.

Parameters

in	*str*	the string to be encoded

Returns

the encoded string

Return values

NULL	an error occurred and errno might be set

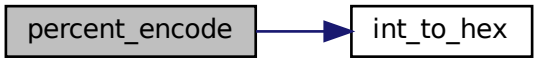
The returned string must be freed using free().

Definition at line 296 of file common.c.

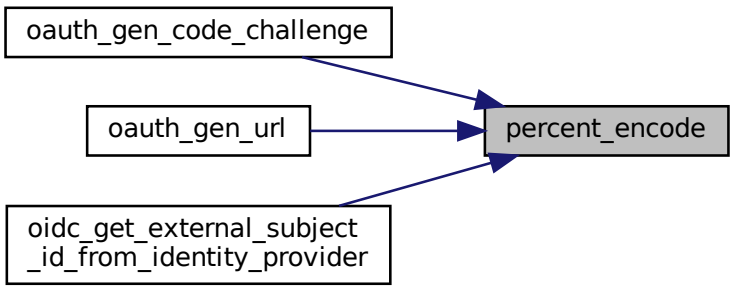
References int_to_hex().

Referenced by oauth_gen_code_challenge(), oauth_gen_url(), and oidc_get_external_subject_id_from_identity_↩provider().

Here is the call graph for this function:



Here is the caller graph for this function:



5.6.2.19 remove_whitespace()

```
void remove_whitespace (
            char * s )
```

Removes all whitespace characters from a string.

Parameters

in,out	*s*	the string

Definition at line 680 of file common.c.

Referenced by cgi_parse_http_cookie().

Here is the caller graph for this function:

5.6.2.20 str_append()

```
char* str_append (
            char * dest,
            const char * src,
            size_t * destlen,
            size_t srclen )
```

Resizes the dest string and appends the src string to the dest string.

Uses realloc() to resize dest to fit dest, src and the terminating null byte, then it appends srclen characters of the src string to the dest string, overwriting the terminating null byte at the end of dest, and then adds a terminating null byte.

Note

dest and src must not be NULL.

Parameters

in,out	*dest*	the destination string
in	*src*	the source string
in,out	*destlen*	pointer where the current length of dest is stored. This function also updates this value. Excluding the terminating null byte. Ignored if NULL.
in	*srclen*	the length of src. This function will use strlen(src), if srclen == 0.

Returns

the new location of dest.

Return values

NULL	an error occurred and errno might be set.

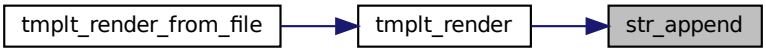
Note

dest is freed by this function if an error occurres.

Definition at line 884 of file common.c.

Referenced by tmplt_render().

Here is the caller graph for this function:



5.6.2.21 uuid_v4_gen()

```
bool uuid_v4_gen (
             char * buffer )
```

Generate a Version 4 UUID according to RFC-4122.

Uses the openssl RAND_bytes function to generate a Version 4 UUID.

Parameters

in,out	*buffer*	A buffer that is at least (UUIDLEN + 1) bytes long.

Return values

true	success
false	failure

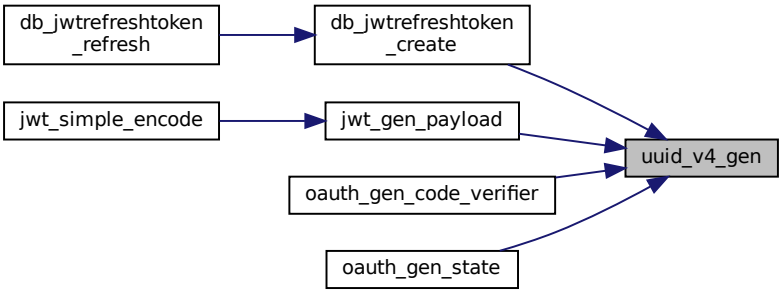
Note

Code from https://gist.github.com/kvelakur/9069c9896577c3040030

Definition at line 35 of file common.c.

Referenced by db_jwtrefreshtoken_create(), jwt_gen_payload(), oauth_gen_code_verifier(), and oauth_gen_↩ state().

Here is the caller graph for this function:



5.6.2.22 writestreamtofile()

```
bool writestreamtofile (
            FILE * stream,
            const char * outpath )
```

Writes a stream to the file located at outpath.

The stream is read until EOF is reached.

Parameters

in	*stream*	the stream to be read from
in	*outpath*	path to the file were the output should be written to

Returns

whether the function was successful

Return values

true	success
false	An error occurred and errno might be set

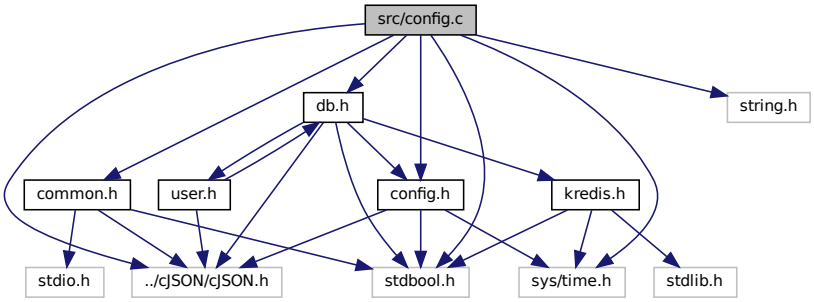
Definition at line 711 of file common.c.

5.7 src/config.c File Reference

Code for parsing the config into a native format.

```
#include "config.h"
#include "common.h"
#include "db.h"
#include "../cJSON/cJSON.h"
#include <sys/time.h>
#include <string.h>
#include <stdbool.h>
```

Include dependency graph for config.c:



Functions

- struct config_s * config_parse (char *errmsg, const char *filepath)

  Parses the config.
- void config_free (struct config_s *config)

  Frees a config.

5.7.1 Detailed Description

Code for parsing the config into a native format.

5.7.1.1 config.json Layout

```
* {
*       "db": { // Database settings
*               "type":"redis",
*               "ip":<string|ip of redis server|can be null>,
*               "port":<number|port of redis server>,
*               "socket":<string|path to unix socket file|can be null>,
*               "use_unix_socket":<bool|whether to use ip+port or socket for the connection>,
*               "timeout": {"seconds":<number|seconds of timeout>, "microseconds": <number|microseconds of tim
*               "auth": [string|redis auth string NOTE: only pre Redis 6.0 requirepass auth string is supporte
*       },
*       "auth": { // Settings for the identity provider
*               "type":"oidc", // OpenID Connect
*               "oauth_authorize_url":<string|oauth authorize endpoint. example: https://gitlab.gwdg.de/oauth/
*               "oauth_token_url":<string|oauth token endpoint. example: https://gitlab.gwdg.de/oauth/token>,
*               "redirect_uri":<string|url of oauthcallback endpoint. must match the redirect uri configured a
*               "scope":<string|space seperated list of oauth scopes. must be equivalent to the scope configur
*               "app_id":<string|the oauth application id>,
*               "app_secret":<string|the oauth application secret>,
*               "oidc_expected_issuer":<string|the expected oidc issuer. `iss` field in id_token. example: htt
*       },
```

```
*       "jwt": { // JSON Web Token settings of the JWT the user gets by the refreshlogin endpoint after authen
*               "secret":<string|JWT secret. must not be bruteforcable!>,
*               "ttl": <number|Time-To-Live of the JWT in seconds>,
*               "leeway": <number|leeway of ttl and ttl_refreshtoken>,
*               "ttl_refreshtoken": <number|Time-To-Live of the JWT Refresh Token in seconds>
*       },
*       "ssh-sign": {
*               "client_ca_private_key": <string|location of the client-ca file. example: ../../client-ca>,
*               "validity": <string|how long a signed key should be valid for. see -V option of ssh-keygen. ex
*       }
* }
*
```

5.7.2 Function Documentation

5.7.2.1 config_free()

```
void config_free (
            struct config_s * config )
```

Frees a config.

Parameters

in	*config*	the config

Definition at line 348 of file config.c.

References config_s::__raw.

5.7.2.2 config_parse()

```
struct config_s* config_parse (
            char * errmsg,
            const char * filepath )
```

Parses the config.

Parameters

out	*errmsg*	buffer for error messages. size: ERRMSGMAXSIZE
in	*filepath*	path to the file containing the config

Returns

the parsed config

Return values

NULL	an error occurred and an error message has been written to errmsg

The returned config must be freed using config_free().

Definition at line 272 of file config.c.

References config_s::__raw, and filetojson().

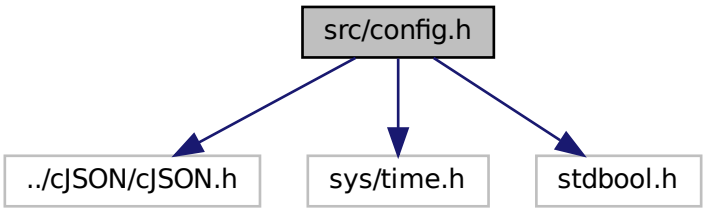
Here is the call graph for this function:

5.8 src/config.h File Reference

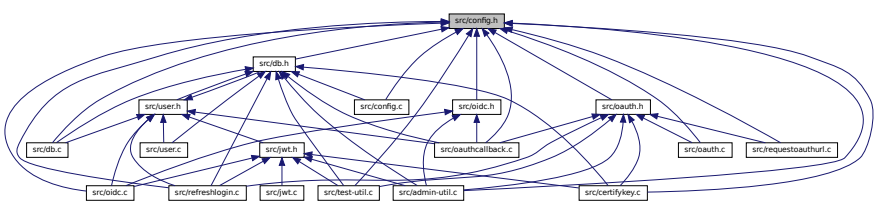
Header file for config.c.

```
#include "../cJSON/cJSON.h"
#include <sys/time.h>
#include <stdbool.h>
```

Include dependency graph for config.h:



This graph shows which files directly or indirectly include this file:



Data Structures

- struct config_db_s

  native representation of config.db
- struct config_auth_s

  native representation of config.auth
- struct config_jwt_s

  native representation of config.jwt
- struct config_certify_s

  native representation of config["ssh-sign"]
- struct config_s

  config struct

Functions

- struct config_s * config_parse (char *errmsg, const char *filepath)

  Parses the config.
- void config_free (struct config_s *config)

  Frees a config.

5.8.1 Detailed Description

Header file for config.c.

5.8.2 Function Documentation

5.8.2.1 config_free()

```
void config_free (
            struct config_s * config )
```

Frees a config.

Parameters

in	*config*	the config

Definition at line 348 of file config.c.

References config_s::__raw.

5.8.2.2 config_parse()

```
struct config_s* config_parse (
            char * errmsg,
            const char * filepath )
```

Parses the config.

Parameters

out	*errmsg*	buffer for error messages. size: ERRMSGMAXSIZE
in	*filepath*	path to the file containing the config

Returns

the parsed config

Return values

NULL	an error occurred and an error message has been written to errmsg

The returned config must be freed using config_free().

Definition at line 272 of file config.c.

References config_s::__raw, and filetojson().

Here is the call graph for this function:

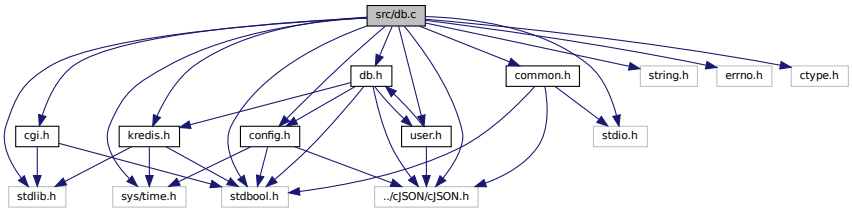
5.9 src/db.c File Reference

Code for interacting with the database on a 'higher'-level.

```
#include "db.h"
#include "common.h"
#include "cgi.h"
#include "kredis.h"
#include "config.h"
#include "user.h"
#include "../cJSON/cJSON.h"
#include <stdio.h>
```

```
#include <stdlib.h>
#include <stdbool.h>
#include <string.h>
#include <errno.h>
#include <ctype.h>
#include <sys/time.h>
```

Include dependency graph for db.c:



Functions

- db_t * db_connect (char *errmsg, const struct config_db_s *dbconfig)

  Connects to a database.
- void db_close (db_t *db)

  Disconnects and frees a database context.
- bool db_user_store (char *errmsg, db_t *db, cJSON *user)

  Stores a user in the database.
- cJSON * db_user_get (char *errmsg, db_t *db, const char *userid)

  Retrieves a user from the database via their userid.
- cJSON * db_users_get_all (char *errmsg, db_t *db)

  Retrieves all users as an array from the database.
- bool db_jwtrefreshtoken_create (char *errmsg, db_t *db, char *token, const user_t *user, long ttl)

  Generates a new JWT refresh token and writes it into the database.
- user_t * db_jwtrefreshtoken_refresh (char *errmsg, db_t *db, char *oldtoken, char *newtoken, long ttl)

  Uses and regenerates a JWT refresh token and updates the database accordingly.

5.9.1 Detailed Description

Code for interacting with the database on a 'higher'-level.

5.9.1.1 Database Layout

type	expire	key	value	comment
string	10min	jwtrefreshtoken↩ :<jwtrefreshtoken>	<userid>	don't delete manually. let it expire
string	never	DB_PREFIX_USE↩ R<userid>	see the JSON Syntax at db_user_store()	
set	never	DB_PREFIX_USERS	<userids>	

All keys have the extra prefix `sshca:`.

5.9.2 Function Documentation

5.9.2.1 db_close()

```
void db_close (
            db_t * db )
```

Disconnects and frees a database context.

Parameters

in,out	*db*	the database context

Definition at line 108 of file db.c.

References config_s::db, and kredis_free().

Referenced by db_connect().

Here is the call graph for this function:

Here is the caller graph for this function:

5.9.2.2 db_connect()

```
db_t* db_connect (
            char * errmsg,
            const struct config_db_s * dbconfig )
```

Connects to a database.

Parameters

out	*errmsg*	buffer for error messages. size: ERRMSGMAXSIZE
in	*dbconfig*	config for connecting to the database

Returns

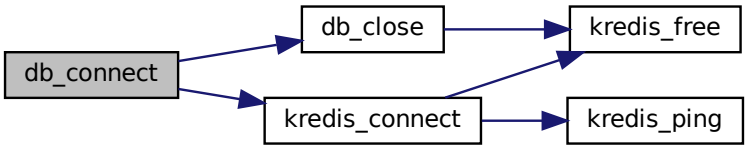
On success a database context is returned and the connection is established. Otherwise NULL is returned, errno might be set and errmsg contains an error message.

The returned pointer must be freed using db_close().

Definition at line 44 of file db.c.

References config_db_s::auth, db_s::c, db_close(), config_db_s::ip, db_s::isconnected, kredis_connect(), config_db_s::port, config_db_s::socket, config_db_s::timeout, config_db_s::type, and config_db_s::use_unix_socket.

Here is the call graph for this function:



5.9.2.3 db_jwtrefreshtoken_create()

```
bool db_jwtrefreshtoken_create (
            char * errmsg,
            db_t * db,
            char * token,
            const user_t * user,
            long ttl )
```

Generates a new JWT refresh token and writes it into the database.

JWT refresh tokens are used for authorizing JWT generation.

A token is always associated with a user. Thus when using a token you can determine for whom the JWT must be generated.

A token can only ever be used once.

A JWT refresh token itself is just a UUIDv4 string.

The database entry will expire after ttl seconds.

See also

db_jwtrefreshtoken_refresh()

Parameters

out	*errmsg*	buffer for error messages. size: ERRMSGMAXSIZE
in,out	*db*	the database context
out	*token*	buffer for storing the JWT refresh token. size: DB_JWTREFRESHTOKEN_BUFSIZE
in	*user*	the user
in	*ttl*	how long the token should be valid in seconds

Returns

whether the function was successful.

Return values

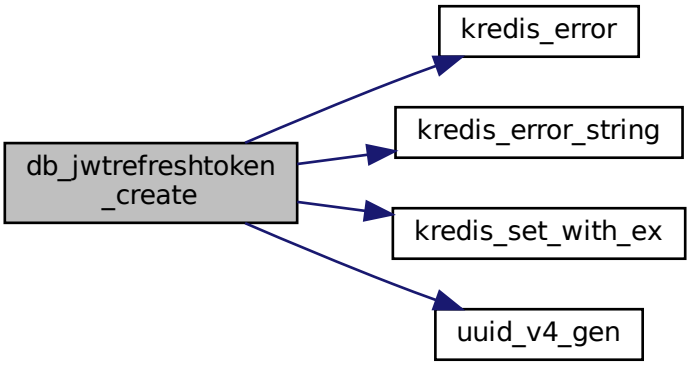
true	success
false	an error occurred and an error message has been written to errmsg

Definition at line 347 of file db.c.

References config_s::db, DB_PREFIX_JWTREFRESHTOKEN, user_s::id, kredis_error(), kredis_error_string(), kredis_set_with_ex(), and uuid_v4_gen().

Referenced by db_jwtrefreshtoken_refresh().

Here is the call graph for this function:



Here is the caller graph for this function:

5.9.2.4 db_jwtrefreshtoken_refresh()

```
user_t* db_jwtrefreshtoken_refresh (
            char * errmsg,
            db_t * db,
            char * oldtoken,
            char * newtoken,
            long ttl )
```

Uses and regenerates a JWT refresh token and updates the database accordingly.

This will invalidate oldtoken. newtoken should be sent back to the user.

See also

db_jwtrefreshtoken_create()

Parameters

out	*errmsg*	buffer for error messages. size: ERRMSGMAXSIZE
in,out	*db*	the database context
out	*oldtoken*	buffer containing the old JWT refresh token about to be used. size: DB_JWTREFRESHTOKEN_BUFSIZE
in	*newtoken*	buffer for storing the new JWT refresh token. size: DB_JWTREFRESHTOKEN_BUFSIZE
in	*ttl*	how long the token should be valid in seconds

Note

oldtoken and newtoken may overlap.

Returns

the user associated with the token

Return values

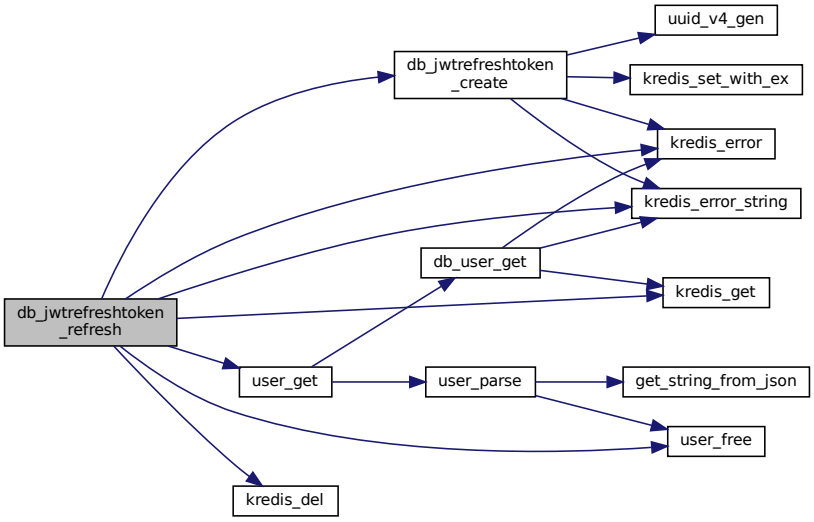
NULL	an error occurred or oldtoken doesn't exist in the database. An error message has been written to errmsg.

The returned user object must be freed using user_free().

Definition at line 405 of file db.c.

References config_s::db, db_jwtrefreshtoken_create(), DB_PREFIX_JWTREFRESHTOKEN, kredis_del(), kredis_error(), kredis_error_string(), kredis_get(), user_free(), and user_get().

Here is the call graph for this function:



5.9.2.5 db_user_get()

```
cJSON* db_user_get (
            char * errmsg,
            db_t * db,
            const char * userid )
```

Retrieves a user from the database via their userid.

See also

db_user_store()

user_get()

Parameters

out	*errmsg*	buffer for error messages. size: ERRMSGMAXSIZE
in,out	*db*	the database context
in	*userid*	the users userid

Returns

the retrieved data as JSON

Return values

NULL	an error occurred or user doesn't exist in the database. An error message has been written to errmsg.

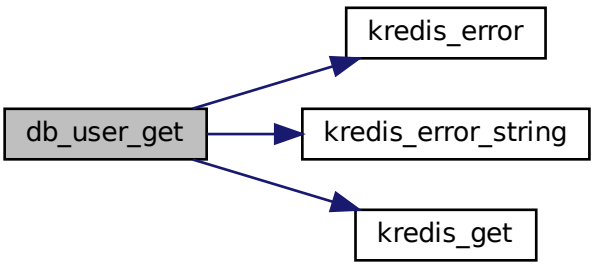
The returned cJSON object must be freed using cJSON_Delete().

Definition at line 225 of file db.c.

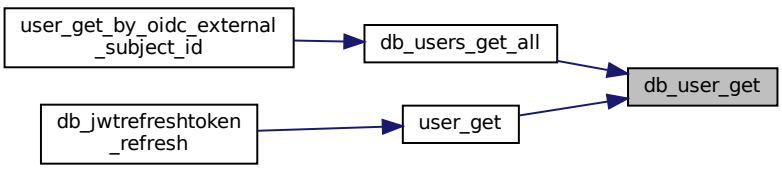
References config_s::db, DB_PREFIX_USER, kredis_error(), kredis_error_string(), and kredis_get().

Referenced by db_users_get_all(), and user_get().

Here is the call graph for this function:



Here is the caller graph for this function:



5.9.2.6 db_user_store()

```
bool db_user_store (
            char * errmsg,
            db_t * db,
            cJSON * user )
```

Stores a user in the database.

The data is stored as JSON and can later be retrieved using db_user_get().

The JSON syntax is as follows:

```
{
        "id":"<userid>",
        "name":"<username>",
        "oidc_external_subject_id":"<external subject id>",
        "state":"<active|blocked>",
        "principals":[
                "<principal>",
                "..."
        ]
}
```

Parameters

out	*errmsg*	buffer for error messages. size: ERRMSGMAXSIZE
in,out	*db*	the database context
in	*user*	the user

Returns

whether the function was successful.

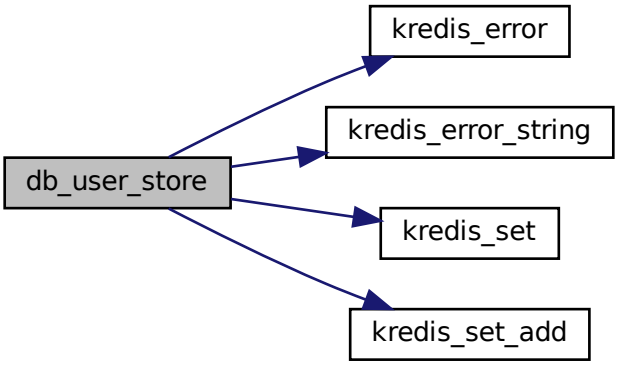
Return values

true	success
false	an error occurred and an error message has been written to errmsg

Definition at line 144 of file db.c.

References config_s::db, DB_PREFIX_USER, DB_PREFIX_USERS, kredis_error(), kredis_error_string(), kredis↩
_set(), and kredis_set_add().

Here is the call graph for this function:



5.9.2.7 db_users_get_all()

```
cJSON* db_users_get_all (
            char * errmsg,
            db_t * db )
```

Retrieves all users as an array from the database.

Parameters

out	*errmsg*	buffer for error messages. size: ERRMSGMAXSIZE
in,out	*db*	the database context

Returns

a cJSON Array containing all the user objects

Return values

NULL	an error occurred. An error message has been written to errmsg.

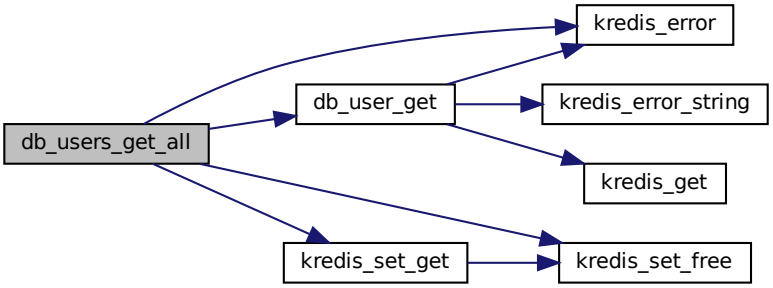
The returned array must be freed using cJSON_Delete().

Definition at line 275 of file db.c.

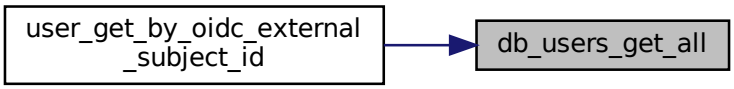
References config_s::db, DB_PREFIX_USERS, db_user_get(), kredis_error(), kredis_set_free(), and kredis_set↩ _get().

Referenced by user_get_by_oidc_external_subject_id().

Here is the call graph for this function:



Here is the caller graph for this function:

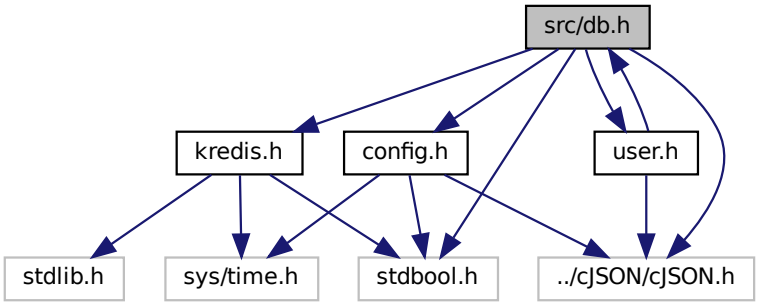


5.10 src/db.h File Reference

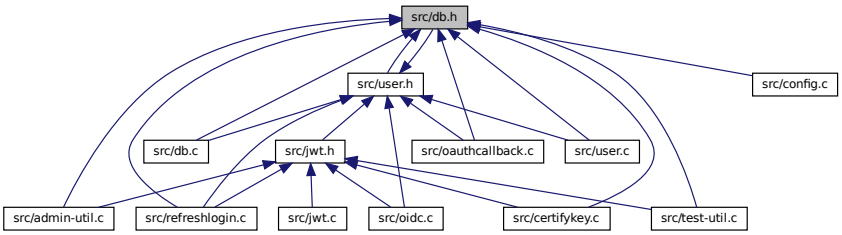
Header file for db.c.

```
#include "kredis.h"
#include "config.h"
#include "user.h"
#include "../cJSON/cJSON.h"
#include <stdbool.h>
```

Include dependency graph for db.h:



This graph shows which files directly or indirectly include this file:



Data Structures

- struct db_s
  database struct

Macros

- #define DB_PREFIX_USER "user:"
  database prefix for user
- #define DB_PREFIX_USERS "users:"
  database prefix for users
- #define DB_PREFIX_JWTREFRESHTOKEN "jwtrefreshtoken:"
  database prefix for jwtrefreshtoken
- #define DB_JWTREFRESHTOKEN_BUFSIZE (UUIDLEN + 1)
  the required buffer size for a jwtrefreshtoken

Typedefs

- typedef struct db_s **db_t**

Functions

- db_t * db_connect (char *errmsg, const struct config_db_s *dbconfig)

  Connects to a database.
- void db_close (db_t *db)

  Disconnects and frees a database context.
- bool db_user_store (char *errmsg, db_t *db, cJSON *user)

  Stores a user in the database.
- cJSON * db_user_get (char *errmsg, db_t *db, const char *userid)

  Retrieves a user from the database via their userid.
- cJSON * db_users_get_all (char *errmsg, db_t *db)

  Retrieves all users as an array from the database.
- bool db_jwtrefreshtoken_create (char *errmsg, db_t *db, char *token, const user_t *user, long ttl)

  Generates a new JWT refresh token and writes it into the database.
- user_t * db_jwtrefreshtoken_refresh (char *errmsg, db_t *db, char *oldtoken, char *newtoken, long ttl)

  Uses and regenerates a JWT refresh token and updates the database accordingly.

5.10.1 Detailed Description

Header file for db.c.

5.10.2 Macro Definition Documentation

5.10.2.1 DB_PREFIX_USERS

```
#define DB_PREFIX_USERS "users:"
```

database prefix for users

TODO remove ':' (requires updating the database accordingly)

Definition at line 28 of file db.h.

5.10.3 Function Documentation

5.10.3.1 db_close()

```
void db_close (
            db_t * db )
```

Disconnects and frees a database context.

Parameters

in,out	*db*	the database context

Definition at line 108 of file db.c.

References config_s::db, and kredis_free().

Referenced by db_connect().

Here is the call graph for this function:

Here is the caller graph for this function:

5.10.3.2 db_connect()

```
db_t* db_connect (
            char * errmsg,
            const struct config_db_s * dbconfig )
```

Connects to a database.

Parameters

out	*errmsg*	buffer for error messages. size: ERRMSGMAXSIZE
in	*dbconfig*	config for connecting to the database

Returns

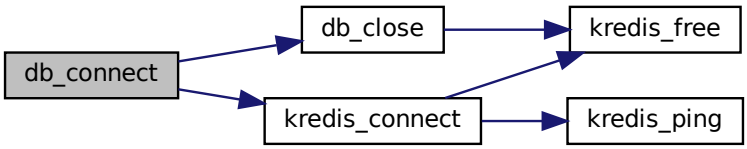
On success a database context is returned and the connection is established. Otherwise NULL is returned, errno might be set and errmsg contains an error message.

The returned pointer must be freed using db_close().

Definition at line 44 of file db.c.

References config_db_s::auth, db_s::c, db_close(), config_db_s::ip, db_s::isconnected, kredis_connect(), config_db_s::port, config_db_s::socket, config_db_s::timeout, config_db_s::type, and config_db_s::use_unix_socket.

Here is the call graph for this function:



5.10.3.3 db_jwtrefreshtoken_create()

```
bool db_jwtrefreshtoken_create (
            char * errmsg,
            db_t * db,
            char * token,
            const user_t * user,
            long ttl )
```

Generates a new JWT refresh token and writes it into the database.

JWT refresh tokens are used for authorizing JWT generation.

A token is always associated with a user. Thus when using a token you can determine for whom the JWT must be generated.

A token can only ever be used once.

A JWT refresh token itself is just a UUIDv4 string.

The database entry will expire after ttl seconds.

See also

db_jwtrefreshtoken_refresh()

Parameters

out	*errmsg*	buffer for error messages. size: ERRMSGMAXSIZE
in,out	*db*	the database context
out	*token*	buffer for storing the JWT refresh token. size: DB_JWTREFRESHTOKEN_BUFSIZE
in	*user*	the user
in	*ttl*	how long the token should be valid in seconds

Returns

whether the function was successful.

Return values

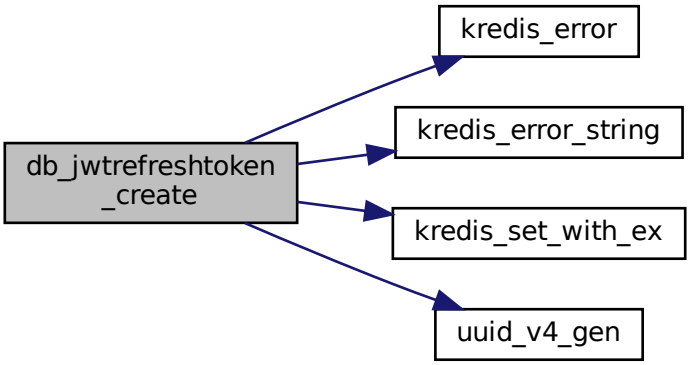
true	success
false	an error occurred and an error message has been written to errmsg

Definition at line 347 of file db.c.

References config_s::db, DB_PREFIX_JWTREFRESHTOKEN, user_s::id, kredis_error(), kredis_error_string(), kredis_set_with_ex(), and uuid_v4_gen().

Referenced by db_jwtrefreshtoken_refresh().

Here is the call graph for this function:



Here is the caller graph for this function:

5.10.3.4 db_jwtrefreshtoken_refresh()

```
user_t* db_jwtrefreshtoken_refresh (
            char * errmsg,
            db_t * db,
            char * oldtoken,
            char * newtoken,
            long ttl )
```

Uses and regenerates a JWT refresh token and updates the database accordingly.

This will invalidate oldtoken. newtoken should be sent back to the user.

See also

db_jwtrefreshtoken_create()

Parameters

out	*errmsg*	buffer for error messages. size: ERRMSGMAXSIZE
in,out	*db*	the database context
out	*oldtoken*	buffer containing the old JWT refresh token about to be used. size: DB_JWTREFRESHTOKEN_BUFSIZE
in	*newtoken*	buffer for storing the new JWT refresh token. size: DB_JWTREFRESHTOKEN_BUFSIZE
in	*ttl*	how long the token should be valid in seconds

Note

oldtoken and newtoken may overlap.

Returns

the user associated with the token

Return values

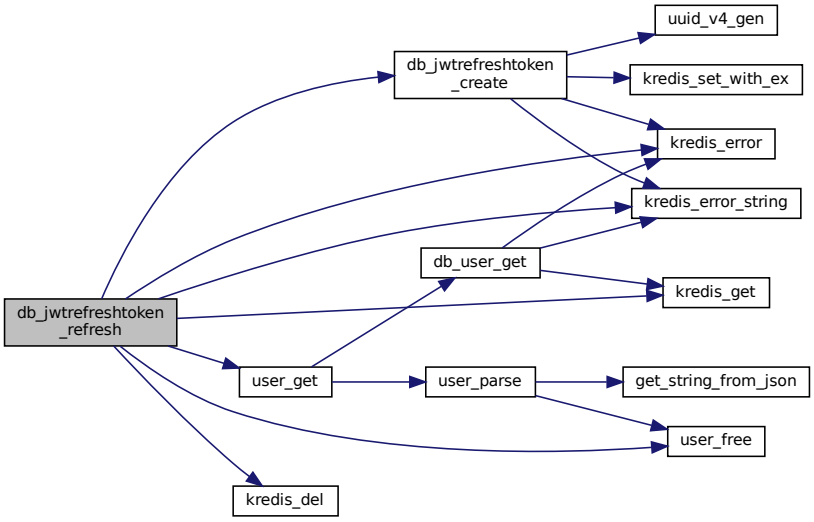
NULL	an error occurred or oldtoken doesn't exist in the database. An error message has been written to errmsg.

The returned user object must be freed using user_free().

Definition at line 405 of file db.c.

References config_s::db, db_jwtrefreshtoken_create(), DB_PREFIX_JWTREFRESHTOKEN, kredis_del(), kredis_error(), kredis_error_string(), kredis_get(), user_free(), and user_get().

Here is the call graph for this function:



5.10.3.5 db_user_get()

```
cJSON* db_user_get (
            char * errmsg,
            db_t * db,
            const char * userid )
```

Retrieves a user from the database via their userid.

See also

db_user_store()

user_get()

Parameters

out	*errmsg*	buffer for error messages. size: ERRMSGMAXSIZE
in,out	*db*	the database context
in	*userid*	the users userid

Returns

the retrieved data as JSON

Return values

NULL	an error occurred or user doesn't exist in the database. An error message has been written to errmsg.

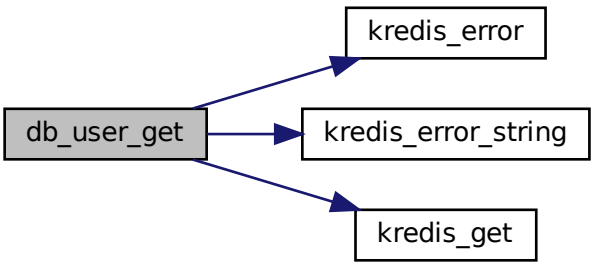
The returned cJSON object must be freed using cJSON_Delete().

Definition at line 225 of file db.c.

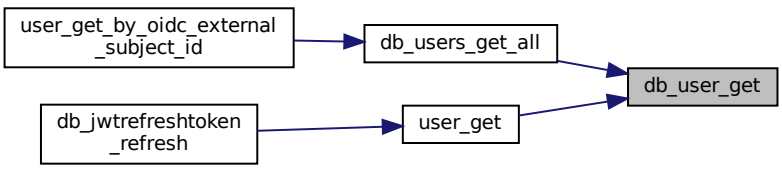
References config_s::db, DB_PREFIX_USER, kredis_error(), kredis_error_string(), and kredis_get().

Referenced by db_users_get_all(), and user_get().

Here is the call graph for this function:



Here is the caller graph for this function:



5.10.3.6 db_user_store()

```
bool db_user_store (
            char * errmsg,
            db_t * db,
            cJSON * user )
```

Stores a user in the database.

The data is stored as JSON and can later be retrieved using db_user_get().

The JSON syntax is as follows:

```
{
        "id":"<userid>",
        "name":"<username>",
        "oidc_external_subject_id":"<external subject id>",
        "state":"<active|blocked>",
        "principals":[
                "<principal>",
                "..."
        ]
}
```

Parameters

out	*errmsg*	buffer for error messages. size: ERRMSGMAXSIZE
in,out	*db*	the database context
in	*user*	the user

Returns

whether the function was successful.

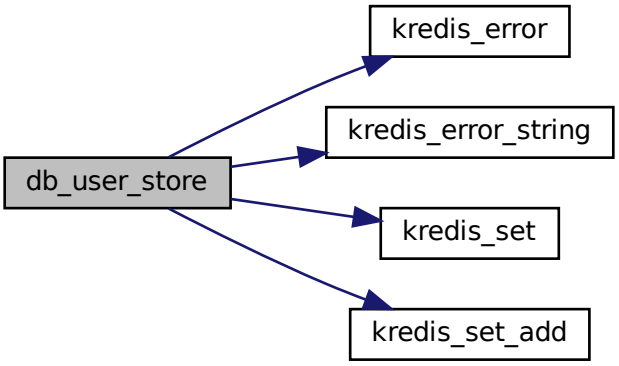
Return values

true	success
false	an error occurred and an error message has been written to errmsg

Definition at line 144 of file db.c.

References config_s::db, DB_PREFIX_USER, DB_PREFIX_USERS, kredis_error(), kredis_error_string(), kredis↩
_set(), and kredis_set_add().

Here is the call graph for this function:



5.10.3.7 db_users_get_all()

```
cJSON* db_users_get_all (
            char * errmsg,
            db_t * db )
```

Retrieves all users as an array from the database.

Parameters

out	*errmsg*	buffer for error messages. size: ERRMSGMAXSIZE
in,out	*db*	the database context

Returns

a cJSON Array containing all the user objects

Return values

NULL	an error occurred. An error message has been written to errmsg.

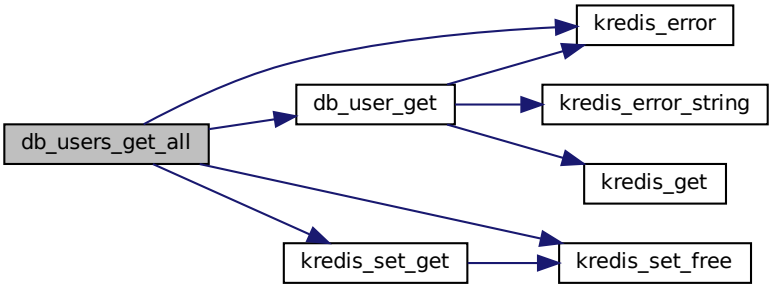
The returned array must be freed using cJSON_Delete().

Definition at line 275 of file db.c.

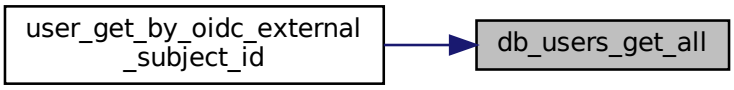
References config_s::db, DB_PREFIX_USERS, db_user_get(), kredis_error(), kredis_set_free(), and kredis_set↩ _get().

Referenced by user_get_by_oidc_external_subject_id().

Here is the call graph for this function:



Here is the caller graph for this function:

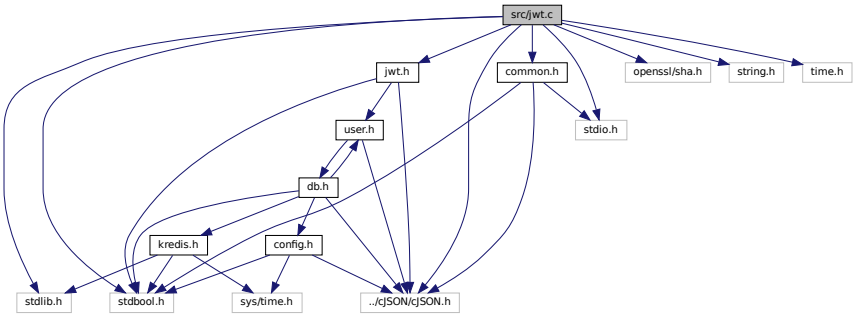


5.11 src/jwt.c File Reference

JSON Web Token.

```
#include "jwt.h"
#include "common.h"
#include "../cJSON/cJSON.h"
#include <openssl/sha.h>
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include <string.h>
#include <time.h>
```

Include dependency graph for jwt.c:



Functions

- char * jwt_simple_encode (char *errmsg, const user_t *user, unsigned long expin, unsigned long leeway, const char *key)

  Wrapper function for creating, encoding and signing a JWT.
- char * jwt_simple_decode (char *errmsg, const char *jwtstr, const char *key)

  Wrapper function for verifying and decoding a JWT and extracting the userid from the sub claim.
- cJSON * jwt_gen_payload (char *errmsg, const user_t *user, unsigned long expin, unsigned long leeway)

  Generates a JWT payload.
- char * jwt_encode_and_sign (const cJSON *payload, const char *key)

  Encodes and signs a JWT.
- bool jwt_verify_encoding (const char *jwtstr)

  Verifies the encoding of a JWT string.
- bool jwt_verify_signature (const char *jwtstr, const char *key)

  Verifies the signature of a JWT string.
- cJSON * jwt_decode (const char *jwtstr)

  Extracts the payload of a JWT string.
- bool jwt_has_expired (const cJSON *payload)

  Tests whether a JWT payload has expired.

5.11.1 Detailed Description

JSON Web Token.

This implementation of JWTs is NOT 100% conforming to RFC7519! See https://datatracker.ietf.org/doc/html/rfc7519#section-8 . For example the algorithm type "none" is NOT implemented! Also the validation is NOT 100% conforming to https://datatracker.ietf.org/doc/html/rfc7519#section-7.2 , as this implementation assumes the algorithm and thus can and does check the signature first.

@TODO Make this implementation 100% conforming to RFC7519.

5.11.2 Function Documentation

5.11.2.1 jwt_decode()

```
cJSON* jwt_decode (
            const char * jwtstr )
```

Extracts the payload of a JWT string.

Parameters

in	*jwtstr*	the JWT as a string

Returns

the payload as JSON

Return values

NULL	an error occurred and errno might be set

Note

You should call jwt_verify_signature() before calling this function.
You should call jwt_has_expired() after calling this function.

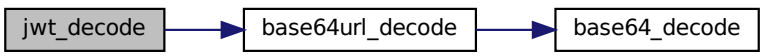
The returned cJSON object must be freed using cJSON_Delete().

Definition at line 465 of file jwt.c.

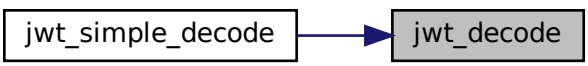
References base64url_decode(), and JWT_HEADER_TYP.

Referenced by jwt_simple_decode().

Here is the call graph for this function:



Here is the caller graph for this function:



5.11.2.2 jwt_encode_and_sign()

```
char* jwt_encode_and_sign (
            const cJSON * payload,
            const char * key )
```

Encodes and signs a JWT.

Uses HS256 as algorithm.

The payload can be generated using jwt_gen_payload().

Note

The payload is taken as is and NOT further modified by this function.

Parameters

in	*payload*	the JWT payload
in	*key*	the key/secret

Returns

a signed JWT as a string

Return values

NULL	an error occurred and errno might be set

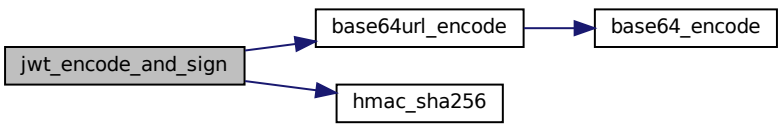
The returned string must be freed using free().

Definition at line 283 of file jwt.c.

References base64url_encode(), hmac_sha256(), JWT_HEADER_ALG, and JWT_HEADER_TYP.

Referenced by jwt_simple_encode().

Here is the call graph for this function:



Here is the caller graph for this function:

5.11.2.3 jwt_gen_payload()

```
cJSON* jwt_gen_payload (
            char * errmsg,
            const user_t * user,
            unsigned long expin,
            unsigned long leeway )
```

Generates a JWT payload.

The payload contains the following claims: iss: JWT_CLAIM_ISS sub: userid iat: current time as nbf: iat - leeway exp: iat + expin + leeway jti: UUIDv4 name: username

See also

https://datatracker.ietf.org/doc/html/rfc7519#section-4.1

Parameters

out	*errmsg*	buffer for error messages. size: ERRMSGMAXSIZE
in	*user*	the user
in	*expin*	number of seconds for which the JWT is valid
in	*leeway*	leeway in seconds to account for clock skew (see https://datatracker.ietf.org/doc/html/rfc7519#section-4.1.4)

Returns

the payload as JSON

Return values

NULL	an error occurred and errno might be set. An error message has been written to errmsg.

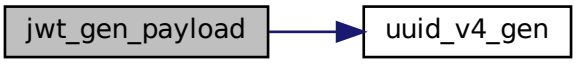
The returned cJSON object must be freed using cJSON_Delete().

Definition at line 168 of file jwt.c.

References user_s::id, JWT_CLAIM_ISS, user_s::name, uuid_v4_gen(), and UUIDLEN.

Referenced by jwt_simple_encode().

Here is the call graph for this function:



Here is the caller graph for this function:

5.11.2.4 jwt_has_expired()

```
bool jwt_has_expired (
            const cJSON * payload )
```

Tests whether a JWT payload has expired.

This function compares the nbf and exp claims in the payload to the current time.

nbf is ignored, if nbf doesn't exist in the payload.

Parameters

in	*payload*	the JWT payload

Returns

whether the payload has expired

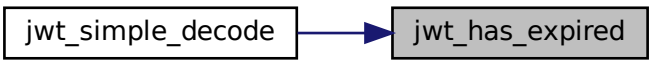
Return values

false	the payload has NOT expired
true	the payload has expired and is thus invalid

Definition at line 566 of file jwt.c.

Referenced by jwt_simple_decode().

Here is the caller graph for this function:



5.11.2.5 jwt_simple_decode()

```
char* jwt_simple_decode (
            char * errmsg,
            const char * jwtstr,
            const char * key )
```

Wrapper function for verifying and decoding a JWT and extracting the userid from the sub claim.

This function calls:

1. jwt_verify_encoding()
2. jwt_verify_signature()
3. jwt_decode()
4. jwt_has_expired()

Parameters

out	*errmsg*	buffer for error messages. size: ERRMSGMAXSIZE
in	*jwtstr*	the JWT as a string
in	*key*	the key/secret

Returns

the sub claim of the payload of the JWT. This should be a userid.

Return values

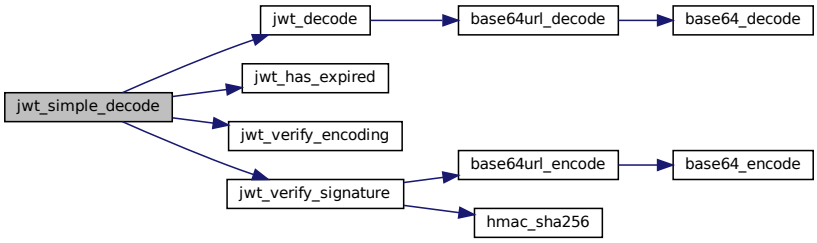
NULL	an error occurred and errno might be set, or jwtstr failed verification. An error message has been written to errmsg.

The returned string must be freed using free().

Definition at line 90 of file jwt.c.

References jwt_decode(), jwt_has_expired(), jwt_verify_encoding(), and jwt_verify_signature().

Here is the call graph for this function:



5.11.2.6 jwt_simple_encode()

```
char* jwt_simple_encode (
            char * errmsg,
            const user_t * user,
            unsigned long expin,
            unsigned long leeway,
            const char * key )
```

Wrapper function for creating, encoding and signing a JWT.

This function calls:

1. jwt_gen_payload()
2. jwt_encode_and_sign()

Parameters

out	*errmsg*	buffer for error messages. size: ERRMSGMAXSIZE
in	*user*	the user
in	*expin*	number of seconds for which the JWT is valid
in	*leeway*	leeway in seconds to account for clock skew (see https://datatracker.ietf.org/doc/html/rfc7519#section-4.1.4)
in	*key*	the key/secret

Returns

a signed JWT as a string

Return values

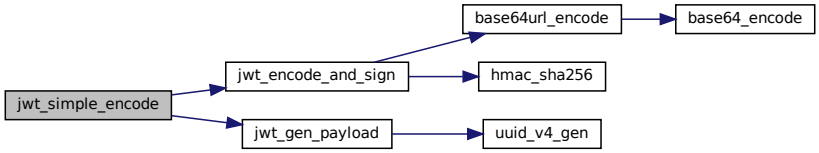
NULL	an error occurred and errno might be set. An error message has been written to errmsg.

The returned string must be freed using free().

Definition at line 45 of file jwt.c.

References jwt_encode_and_sign(), and jwt_gen_payload().

Here is the call graph for this function:



5.11.2.7 jwt_verify_encoding()

```
bool jwt_verify_encoding (
            const char * jwtstr )
```

Verifies the encoding of a JWT string.

This function is a really basic encoding verification. It only checks whether all characters are valid base64url characters, the number of dots and the minimal distance between the dots. That's it!

Parameters

in	*jwtstr*	the JWT as a string

Returns

whether jwtstr is encoded correctly

Return values

true	jwtstr might be a valid JWT string
false	jwtstr is not a valid JWT string

Definition at line 363 of file jwt.c.

Referenced by jwt_simple_decode().

Here is the caller graph for this function:

5.11.2.8 jwt_verify_signature()

```
bool jwt_verify_signature (
            const char * jwtstr,
            const char * key )
```

Verifies the signature of a JWT string.

Parameters

in	*jwtstr*	the JWT as a string
in	*key*	the key to be used for verification

Returns

whether jwtstr is signed by key

Return values

true	jwtstr was signed with key
false	jwtstr was NOT signed with key

Note

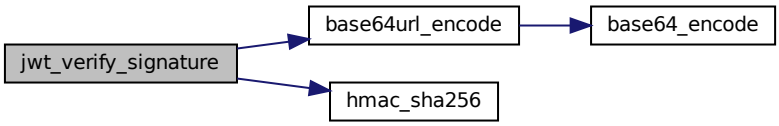
You should call jwt_verify_encoding() before calling this function.

Definition at line 408 of file jwt.c.

References base64url_encode(), and hmac_sha256().

Referenced by jwt_simple_decode().

Here is the call graph for this function:



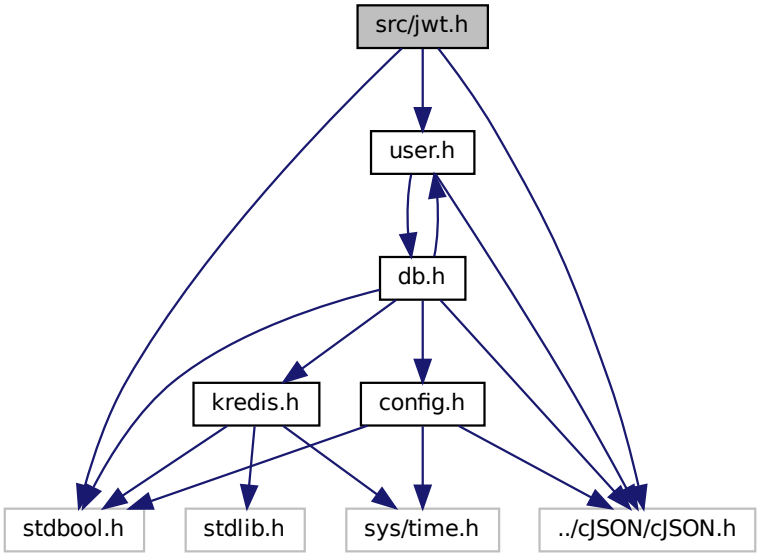
Here is the caller graph for this function:

5.12 src/jwt.h File Reference

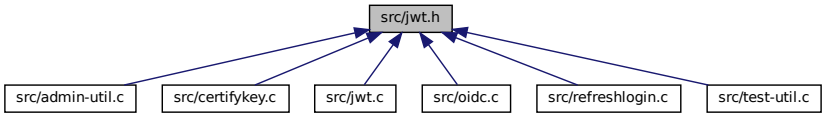
Header file for jwt.c.

```
#include "user.h"
#include "../cJSON/cJSON.h"
#include <stdbool.h>
```

Include dependency graph for jwt.h:



This graph shows which files directly or indirectly include this file:



Macros

- #define JWT_HEADER_TYP "JWT"

  JWT Type.
- #define JWT_HEADER_ALG "HS256"

  used JWT Algorithm
- #define JWT_CLAIM_ISS "SSHCA"

  JWT Issuer.

Functions

- char * jwt_simple_encode (char *errmsg, const user_t *user, unsigned long expin, unsigned long leeway, const char *key)

  Wrapper function for creating, encoding and signing a JWT.
- char * jwt_simple_decode (char *errmsg, const char *jwtstr, const char *key)

  Wrapper function for verifying and decoding a JWT and extracting the userid from the sub claim.
- cJSON * jwt_gen_payload (char *errmsg, const user_t *user, unsigned long expin, unsigned long leeway)

  Generates a JWT payload.
- char * jwt_encode_and_sign (const cJSON *payload, const char *key)

  Encodes and signs a JWT.
- bool jwt_verify_encoding (const char *jwtstr)

  Verifies the encoding of a JWT string.
- bool jwt_verify_signature (const char *jwtstr, const char *key)

  Verifies the signature of a JWT string.
- cJSON * jwt_decode (const char *jwtstr)

  Extracts the payload of a JWT string.
- bool jwt_has_expired (const cJSON *payload)

  Tests whether a JWT payload has expired.

5.12.1 Detailed Description

Header file for jwt.c.

5.12.2 Function Documentation

5.12.2.1 jwt_decode()

```
cJSON* jwt_decode (
            const char * jwtstr )
```

Extracts the payload of a JWT string.

Parameters

in	*jwtstr*	the JWT as a string

Returns

the payload as JSON

Return values

NULL	an error occurred and errno might be set

Note

You should call jwt_verify_signature() before calling this function.

You should call jwt_has_expired() after calling this function.

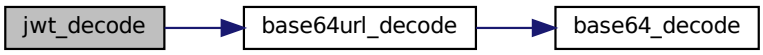
The returned cJSON object must be freed using cJSON_Delete().

Definition at line 465 of file jwt.c.

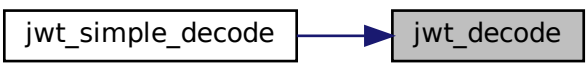
References base64url_decode(), and JWT_HEADER_TYP.

Referenced by jwt_simple_decode().

Here is the call graph for this function:



Here is the caller graph for this function:



5.12.2.2 jwt_encode_and_sign()

```
char* jwt_encode_and_sign (
            const cJSON * payload,
            const char * key )
```

Encodes and signs a JWT.

Uses HS256 as algorithm.

The payload can be generated using jwt_gen_payload().

Note

The payload is taken as is and NOT further modified by this function.

Parameters

in	*payload*	the JWT payload
in	*key*	the key/secret

Returns

a signed JWT as a string

Return values

NULL	an error occurred and errno might be set

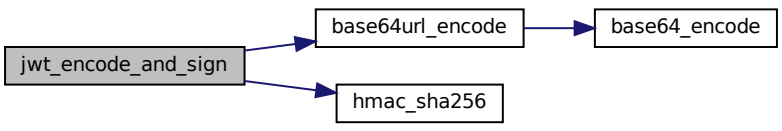
The returned string must be freed using free().

Definition at line 283 of file jwt.c.

References base64url_encode(), hmac_sha256(), JWT_HEADER_ALG, and JWT_HEADER_TYP.

Referenced by jwt_simple_encode().

Here is the call graph for this function:



Here is the caller graph for this function:

5.12.2.3 jwt_gen_payload()

```
cJSON* jwt_gen_payload (
            char * errmsg,
            const user_t * user,
            unsigned long expin,
            unsigned long leeway )
```

Generates a JWT payload.

The payload contains the following claims: iss: JWT_CLAIM_ISS sub: userid iat: current time as nbf: iat - leeway exp: iat + expin + leeway jti: UUIDv4 name: username

See also

https://datatracker.ietf.org/doc/html/rfc7519#section-4.1

Parameters

out	*errmsg*	buffer for error messages. size: ERRMSGMAXSIZE
in	*user*	the user
in	*expin*	number of seconds for which the JWT is valid
in	*leeway*	leeway in seconds to account for clock skew (see https://datatracker.ietf.org/doc/html/rfc7519#section-4.1.4)

Returns

the payload as JSON

Return values

NULL	an error occurred and errno might be set. An error message has been written to errmsg.

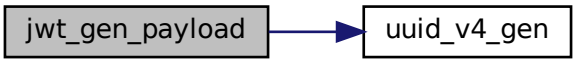
The returned cJSON object must be freed using cJSON_Delete().

Definition at line 168 of file jwt.c.

References user_s::id, JWT_CLAIM_ISS, user_s::name, uuid_v4_gen(), and UUIDLEN.

Referenced by jwt_simple_encode().

Here is the call graph for this function:



Here is the caller graph for this function:

5.12.2.4 jwt_has_expired()

```
bool jwt_has_expired (
            const cJSON * payload )
```

Tests whether a JWT payload has expired.

This function compares the nbf and exp claims in the payload to the current time.

nbf is ignored, if nbf doesn't exist in the payload.

Parameters

in	*payload*	the JWT payload

Returns

whether the payload has expired

Return values

false	the payload has NOT expired
true	the payload has expired and is thus invalid

Definition at line 566 of file jwt.c.

Referenced by jwt_simple_decode().

Here is the caller graph for this function:

5.12.2.5 jwt_simple_decode()

```
char* jwt_simple_decode (
            char * errmsg,
            const char * jwtstr,
            const char * key )
```

Wrapper function for verifying and decoding a JWT and extracting the userid from the sub claim.

This function calls:

1. jwt_verify_encoding()
2. jwt_verify_signature()
3. jwt_decode()
4. jwt_has_expired()

Parameters

out	*errmsg*	buffer for error messages. size: ERRMSGMAXSIZE
in	*jwtstr*	the JWT as a string
in	*key*	the key/secret

Returns

the sub claim of the payload of the JWT. This should be a userid.

Return values

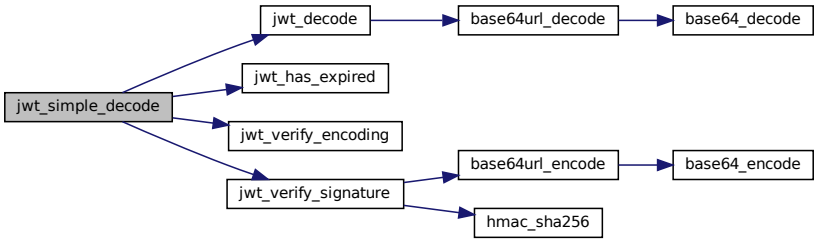
NULL	an error occurred and errno might be set, or jwtstr failed verification. An error message has been written to errmsg.

The returned string must be freed using free().

Definition at line 90 of file jwt.c.

References jwt_decode(), jwt_has_expired(), jwt_verify_encoding(), and jwt_verify_signature().

Here is the call graph for this function:



5.12.2.6 jwt_simple_encode()

```
char* jwt_simple_encode (
            char * errmsg,
            const user_t * user,
            unsigned long expin,
            unsigned long leeway,
            const char * key )
```

Wrapper function for creating, encoding and signing a JWT.

This function calls:

1. jwt_gen_payload()
2. jwt_encode_and_sign()

Parameters

out	*errmsg*	buffer for error messages. size: ERRMSGMAXSIZE
in	*user*	the user
in	*expin*	number of seconds for which the JWT is valid
in	*leeway*	leeway in seconds to account for clock skew (see https://datatracker.ietf.org/doc/html/rfc7519#section-4.1.4)
in	*key*	the key/secret

Returns

a signed JWT as a string

Return values

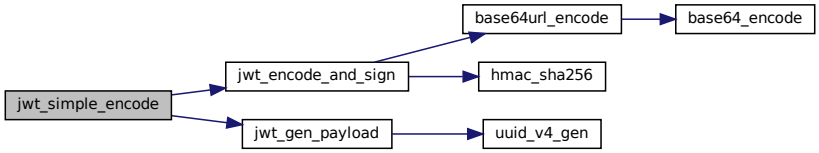
NULL	an error occurred and errno might be set. An error message has been written to errmsg.

The returned string must be freed using free().

Definition at line 45 of file jwt.c.

References jwt_encode_and_sign(), and jwt_gen_payload().

Here is the call graph for this function:



5.12.2.7 jwt_verify_encoding()

```
bool jwt_verify_encoding (
            const char * jwtstr )
```

Verifies the encoding of a JWT string.

This function is a really basic encoding verification. It only checks whether all characters are valid base64url characters, the number of dots and the minimal distance between the dots. That's it!

Parameters

in	*jwtstr*	the JWT as a string

Returns

whether jwtstr is encoded correctly

Return values

true	jwtstr might be a valid JWT string
false	jwtstr is not a valid JWT string

Definition at line 363 of file jwt.c.

Referenced by jwt_simple_decode().

Here is the caller graph for this function:

5.12.2.8 jwt_verify_signature()

```
bool jwt_verify_signature (
            const char * jwtstr,
            const char * key )
```

Verifies the signature of a JWT string.

Parameters

in	*jwtstr*	the JWT as a string
in	*key*	the key to be used for verification

Returns

whether jwtstr is signed by key

Return values

true	jwtstr was signed with key
false	jwtstr was NOT signed with key

Note

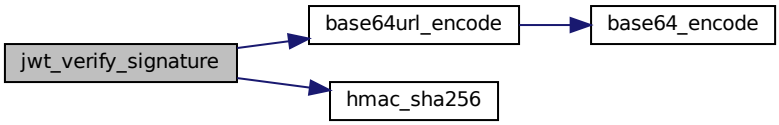
You should call jwt_verify_encoding() before calling this function.

Definition at line 408 of file jwt.c.

References base64url_encode(), and hmac_sha256().

Referenced by jwt_simple_decode().

Here is the call graph for this function:



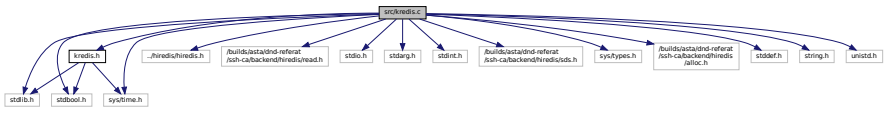
Here is the caller graph for this function:

5.13 src/kredis.c File Reference

Simple ABI for hiredis.

```
#include "kredis.h"
#include "../hiredis/hiredis.h"
#include <stdlib.h>
#include <string.h>
#include <stdbool.h>
#include <unistd.h>
```

Include dependency graph for kredis.c:



Data Structures

- struct kredis_s

Functions

- void kredis_free (kredis_t *k)

  Disconnects and frees a kredis_t object.
- kredis_t * kredis_connect (const char *ip, int port, bool is_unix_socket, struct timeval timeout, const char *auth)

  Connects to a redis server.
- bool kredis_ping (kredis_t *k)

  Ping the redis server.
- int kredis_error (kredis_t *k)

  Gets the current error code as an integer.
- const char * kredis_error_string (kredis_t *k)

  Gets the current error code as a string.
- char * kredis_get (kredis_t *k, const char *key)

  Native Redis GET Command.
- bool kredis_set (kredis_t *k, const char *key, const char *value)

  Native Redis SET Command.
- bool kredis_set_with_ex (kredis_t *k, const char *key, const char *value, long ex)

  Native Redis SET Command with EX set.
- bool kredis_del (kredis_t *k, const char *key)

  Native Redis DEL Command.
- bool kredis_set_add (kredis_t *k, const char *key, const char *member)

  Native Redis SADD Command.
- long long kredis_set_num (kredis_t *k, const char *key)

  Native Redis SCARD Command.
- bool kredis_set_ismember (kredis_t *k, const char *key, const char *member)

  Native Redis SISMEMBER Command.
- bool kredis_set_rem (kredis_t *k, const char *key, const char *member)

  Native Redis SREM Command.
- char ** kredis_set_get (kredis_t *k, const char *key, size_t *nummembers)

  Native Redis SMEMBERS Command.
- void kredis_set_free (char **elements, size_t nummembers)

  Frees the returned array from kredis_set_get()
- bool kredis_hash_set (kredis_t *k, const char *key, const char *field, const char *value)

  Native Redis HSET Command.
- bool kredis_hash_exists (kredis_t *k, const char *key, const char *field)

  Native Redis HEXISTS Command.
- char * kredis_hash_get (kredis_t *k, const char *key, const char *field)

  Native Redis HGET Command.

5.13.1 Detailed Description

Simple ABI for hiredis.

See also

https://github.com/redis/hiredis

Note

The redis server must be at least version 2.6.12.

5.13.2 Function Documentation

5.13.2.1 kredis_connect()

```
kredis_t* kredis_connect (
            const char * ip,
            int port,
            bool is_unix_socket,
            struct timeval timeout,
            const char * auth )
```

Connects to a redis server.

Parameters

in	*ip*	The IP Address of the server or the path to a unix socket. If NULL set by default to 127.0.0.1.
in	*port*	The port of the redis server. Note: The default redis port is 6379.
in	*is_unix_socket*	Whether ip is a path to a unix socket. port is ignored if set.
in	*timeout*	Sets the connection timeout.
in	*auth*	The redis server password specified in the server config by requirepass. Ignored if equal to NULL.

Returns

On success a kredis context is returned. Otherwise NULL is returned and errno might be set.

The returned kredis context must be freed using kredis_free().

Note

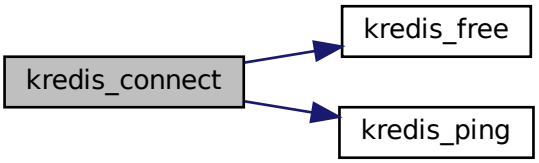
Only the old authentication method prior to Redis 6.0 is supported.

Definition at line 81 of file kredis.c.

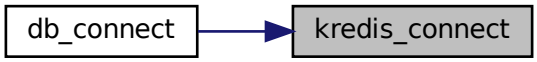
References kredis_free(), kredis_ping(), and kredis_s::rc.

Referenced by db_connect().

Here is the call graph for this function:



Here is the caller graph for this function:



5.13.2.2 kredis_del()

```
bool kredis_del (
            kredis_t * k,
            const char * key )
```

Native Redis DEL Command.

Removes the specified keys. A key is ignored if it does not exist.

See also

https://redis.io/commands/del

Parameters

in,out	*k*	the kredis context
in	*key*	the key

Returns

whether the command was successful.

Return values

false	failed
true	success

Note

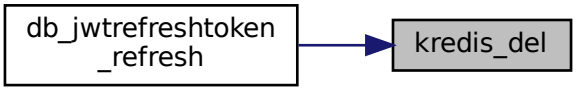
Check for errors using kredis_error() after calling this function.

Definition at line 383 of file kredis.c.

References KREDIS_PREFIX, and kredis_s::rc.

Referenced by db_jwtrefreshtoken_refresh().

Here is the caller graph for this function:



5.13.2.3 kredis_error()

```
int kredis_error (
            kredis_t * k )
```

Gets the current error code as an integer.

In case an error actually happend, you can use kredis_error_string() to get the error message as a string.

Parameters

in	*k*	the kredis context

Returns

the error code as an integer

Return values

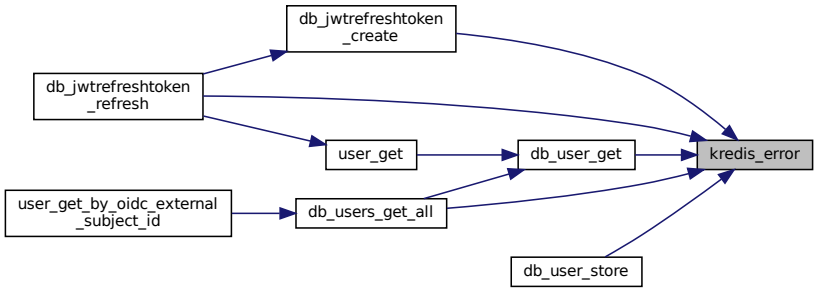
0	indicates no error

Definition at line 192 of file kredis.c.

References kredis_s::rc.

Referenced by db_jwtrefreshtoken_create(), db_jwtrefreshtoken_refresh(), db_user_get(), db_user_store(), and db_users_get_all().

Here is the caller graph for this function:



5.13.2.4 kredis_error_string()

```
const char* kredis_error_string (
            kredis_t * k )
```

Gets the current error code as a string.

This function cannot be used for checking whether an error actually occurred, use kredis_error() instead.

An empty string is returned if no error happend.

Parameters

in	*k*	the kredis context

Returns

the error message

Return values

NULL	invalid argument or invalid redisContext

Note

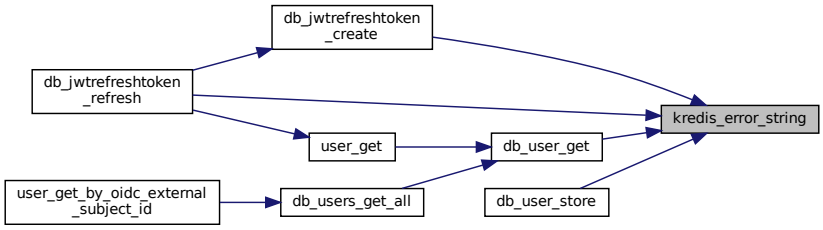
The returned string MUST NOT be freed.

Definition at line 213 of file kredis.c.

References kredis_s::rc.

Referenced by db_jwtrefreshtoken_create(), db_jwtrefreshtoken_refresh(), db_user_get(), and db_user_store().

Here is the caller graph for this function:



5.13.2.5 kredis_free()

```
void kredis_free (
            kredis_t * k )
```

Disconnects and frees a kredis_t object.

Parameters

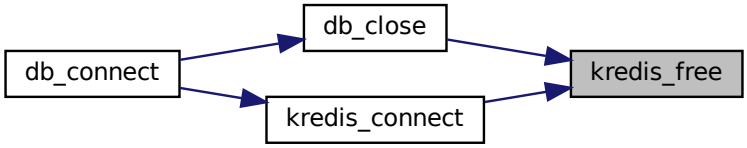
in,out	*k*	the kredis context

Definition at line 57 of file kredis.c.

References kredis_s::rc.

Referenced by db_close(), and kredis_connect().

Here is the caller graph for this function:



5.13.2.6 kredis_get()

```
char* kredis_get (
            kredis_t * k,
            const char * key )
```

Native Redis GET Command.

Get the value of key.

The returned string must be freed using free().

See also

https://redis.io/commands/get

Parameters

in,out	*k*	the kredis context
in	*key*	the key

Returns

the value of key

Return values

NULL	key doesn't exists or isn't a string

Note

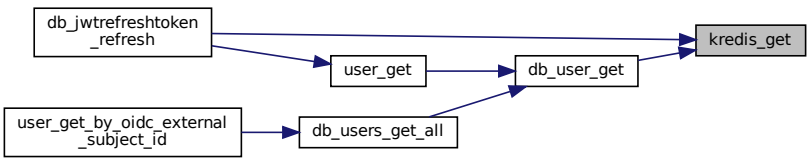
Check for errors using kredis_error() after calling this function.

Definition at line 238 of file kredis.c.

References KREDIS_PREFIX, and kredis_s::rc.

Referenced by db_jwtrefreshtoken_refresh(), and db_user_get().

Here is the caller graph for this function:



5.13.2.7 kredis_hash_exists()

```
bool kredis_hash_exists (
            kredis_t * k,
            const char * key,
            const char * field )
```

Native Redis HEXISTS Command.

Returns if field is an existing field in the hash stored at key.

See also

https://redis.io/commands/hexists

Parameters

in,out	*k*	the kredis context
in	*key*	the key
in	*field*	the field

Returns

whether the field exists in the hash

Return values

true	the hash contains field
false	the hash does not contain field, key does not exist, or other failure

Note

Check for errors using kredis_error() after calling this function.

Definition at line 719 of file kredis.c.

References KREDIS_PREFIX, and kredis_s::rc.

5.13.2.8 kredis_hash_get()

```
char* kredis_hash_get (
            kredis_t * k,
            const char * key,
            const char * field )
```

Native Redis HGET Command.

Returns the value associated with field in the hash stored at key.

The returned string must be freed using free().

See also

https://redis.io/commands/hget

Parameters

in,out	*k*	the kredis context
in	*key*	the key
in	*field*	the field

Returns

the value

Return values

NULL	something failed, errno might be set

Note

Check for errors using kredis_error() after calling this function.

Definition at line 759 of file kredis.c.

References KREDIS_PREFIX, and kredis_s::rc.

5.13.2.9 kredis_hash_set()

```
bool kredis_hash_set (
            kredis_t * k,
            const char * key,
            const char * field,
            const char * value )
```

Native Redis HSET Command.

Sets field in the hash stored at key to value. If key does not exist, a new key holding a hash is created. If field already exists in the hash, it is overwritten.

See also

https://redis.io/commands/hset

Parameters

in,out	*k*	the kredis context
in	*key*	the key
in	*field*	the field
in	*value*	the value

Returns

whether the command was successful.

Return values

false	failed
true	success

Note

Check for errors using kredis_error() after calling this function.

Definition at line 681 of file kredis.c.

References KREDIS_PREFIX, and kredis_s::rc.

5.13.2.10 kredis_ping()

```
bool kredis_ping (
            kredis_t * k )
```

Ping the redis server.

Parameters

in,out	*k*	the kredis context

Returns

whether the ping was successful.

Return values

false	failed
true	success

Note

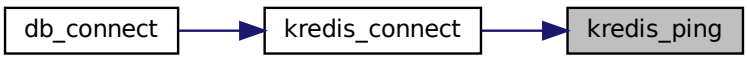
Check for errors using kredis_error() on failure.

Definition at line 160 of file kredis.c.

References kredis_s::rc.

Referenced by kredis_connect().

Here is the caller graph for this function:



5.13.2.11 kredis_set()

```
bool kredis_set (
            kredis_t * k,
            const char * key,
            const char * value )
```

Native Redis SET Command.

Set key to hold the string value. If key already holds a value, it is overwritten, regardless of its type. Any previous time to live associated with the key is discarded on successful SET operation.

See also

https://redis.io/commands/set

Parameters

in,out	*k*	the kredis context
in	*key*	the key
in	*value*	the value

Returns

whether the command was successful.

Return values

false	failed
true	success

Note

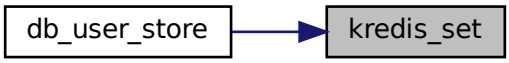
Check for errors using kredis_error() after calling this function.

Definition at line 284 of file kredis.c.

References KREDIS_PREFIX, and kredis_s::rc.

Referenced by db_user_store().

Here is the caller graph for this function:



5.13.2.12 kredis_set_add()

```
bool kredis_set_add (
            kredis_t * k,
            const char * key,
            const char * member )
```

Native Redis SADD Command.

Add the specified member to the set stored at key. Specified members that are already a member of this set are ignored. If key does not exist, a new set is created before adding the specified member.

An error occures when the value stored at key is not a set.

See also

https://redis.io/commands/sadd

Parameters

in,out	*k*	the kredis context
in	*key*	the key
in	*member*	the member to be added

Returns

whether the command was successful.

Return values

false	failed
true	success

Note

Check for errors using kredis_error() after calling this function.

Definition at line 426 of file kredis.c.

References KREDIS_PREFIX, and kredis_s::rc.

Referenced by db_user_store().

Here is the caller graph for this function:

5.13.2.13 kredis_set_free()

```
void kredis_set_free (
            char ** elements,
            size_t nummembers )
```

Frees the returned array from kredis_set_get()

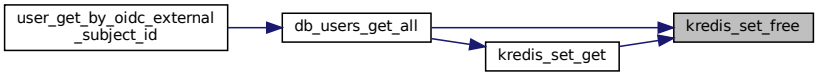
Parameters

in,out	*elements*	the array
in	*nummembers*	the size of the array

Definition at line 650 of file kredis.c.

Referenced by db_users_get_all(), and kredis_set_get().

Here is the caller graph for this function:



5.13.2.14 kredis_set_get()

```
char** kredis_set_get (
            kredis_t * k,
            const char * key,
            size_t * nummembers )
```

Native Redis SMEMBERS Command.

Returns all the members of the set value stored at key.

The returned array must be freed using kredis_set_free().

See also

https://redis.io/commands/smembers

Parameters

in,out	*k*	the kredis context
in	*key*	the key
out	*nummembers*	the number of elements that were read and thus the size of the returned array

Returns

an array of strings

Return values

NULL	something failed, errno might be set

Note

Check for errors using kredis_error() after calling this function.

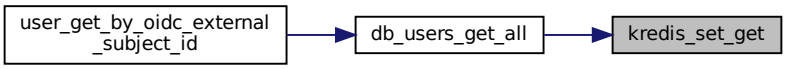
Definition at line 585 of file kredis.c.

References KREDIS_PREFIX, kredis_set_free(), and kredis_s::rc.

Referenced by db_users_get_all().

Here is the call graph for this function:

Here is the caller graph for this function:



5.13.2.15 kredis_set_ismember()

```
bool kredis_set_ismember (
            kredis_t * k,
            const char * key,
            const char * member )
```

Native Redis SISMEMBER Command.

Returns if member is a member of the set stored at key.

See also

https://redis.io/commands/sismember

Parameters

in,out	*k*	the kredis context
in	*key*	the key
in	*member*	the member

Returns

whether member is a member of the set

Return values

true	the element is a member of the set
false	the element is not a member of the set, key does not exist, or other failure

Note

Check for errors using kredis_error() after calling this function.

Definition at line 503 of file kredis.c.

References KREDIS_PREFIX, and kredis_s::rc.

5.13.2.16 kredis_set_num()

```
long long kredis_set_num (
            kredis_t * k,
            const char * key )
```

Native Redis SCARD Command.

Returns the set cardinality (number of elements) of the set stored at key.

See also

https://redis.io/commands/scard

Parameters

in,out	*k*	the kredis context
in	*key*	the key

Returns

the set cardinality

Return values

0	failed or there are no elements in the set

Note

Check for errors using kredis_error() after calling this function.

Definition at line 463 of file kredis.c.

References KREDIS_PREFIX, and kredis_s::rc.

5.13.2.17 kredis_set_rem()

```
bool kredis_set_rem (
            kredis_t * k,
            const char * key,
            const char * member )
```

Native Redis SREM Command.

Remove the specified member from the set stored at key. Specified members that are not a member of this set are ignored. If key does not exist, it is treated as an empty set.

An error occures when the value stored at key is not a set.

See also

https://redis.io/commands/srem

Parameters

in,out	*k*	the kredis context
in	*key*	the key
in	*member*	the member to be removed

Returns

whether the command was successful.

Return values

false	failed
true	success

Note

Check for errors using kredis_error() after calling this function.

Definition at line 546 of file kredis.c.

References KREDIS_PREFIX, and kredis_s::rc.

5.13.2.18 kredis_set_with_ex()

```
bool kredis_set_with_ex (
            kredis_t * k,
            const char * key,
            const char * value,
            long ex )
```

Native Redis SET Command with EX set.

Set key to hold the string value. If key already holds a value, it is overwritten, regardless of its type. Any previous time to live associated with the key is discarded on successful SET operation.

See also

https://redis.io/commands/set

Parameters

in,out	*k*	the kredis context
in	*key*	the key
in	*value*	the value
in	*ex*	expire time, in seconds

Returns

whether the command was successful.

Return values

false	failed
true	success

Note

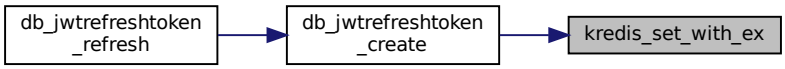
Check for errors using kredis_error() after calling this function.

Definition at line 333 of file kredis.c.

References KREDIS_PREFIX, and kredis_s::rc.

Referenced by db_jwtrefreshtoken_create().

Here is the caller graph for this function:

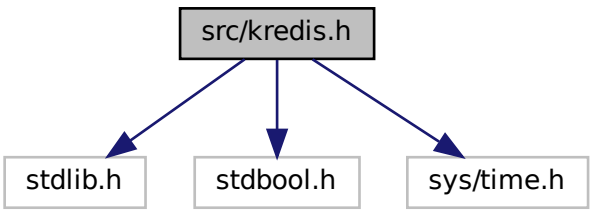


5.14 src/kredis.h File Reference

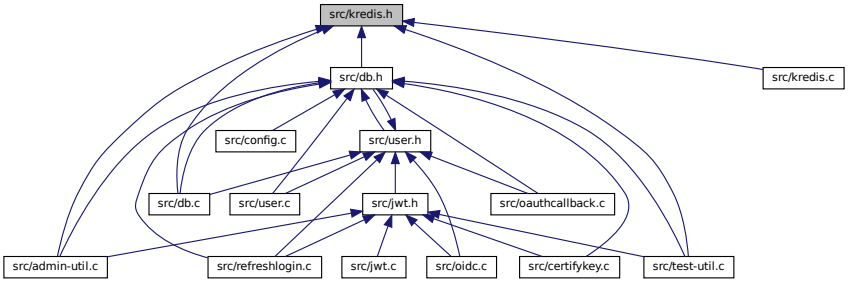
Header file for kredis.c.

```
#include <stdlib.h>
#include <stdbool.h>
#include <sys/time.h>
```

Include dependency graph for kredis.h:



This graph shows which files directly or indirectly include this file:



Macros

- #define KREDIS_PREFIX "sshca:"

  Extra prefix for every key.

Typedefs

- typedef struct kredis_s kredis_t

  A kredis context.

Functions

- kredis_t * kredis_connect (const char *ip, int port, bool is_unix_socket, struct timeval timeout, const char *auth)

  Connects to a redis server.
- void kredis_free (kredis_t *k)

  Disconnects and frees a kredis_t object.
- bool kredis_ping (kredis_t *k)

  Ping the redis server.
- int kredis_error (kredis_t *k)

  Gets the current error code as an integer.
- const char * kredis_error_string (kredis_t *k)

  Gets the current error code as a string.
- char * kredis_get (kredis_t *k, const char *key)

  Native Redis GET Command.
- bool kredis_set (kredis_t *k, const char *key, const char *value)

  Native Redis SET Command.
- bool kredis_set_with_ex (kredis_t *k, const char *key, const char *value, long ex)

  Native Redis SET Command with EX set.
- bool kredis_del (kredis_t *k, const char *key)

  Native Redis DEL Command.
- bool kredis_set_add (kredis_t *k, const char *key, const char *member)

  Native Redis SADD Command.
- long long kredis_set_num (kredis_t *k, const char *key)

  Native Redis SCARD Command.

- bool kredis_set_ismember (kredis_t *k, const char *key, const char *member)

  Native Redis SISMEMBER Command.
- char ** kredis_set_get (kredis_t *k, const char *key, size_t *nummembers)

  Native Redis SMEMBERS Command.
- bool kredis_set_rem (kredis_t *k, const char *key, const char *member)

  Native Redis SREM Command.
- void kredis_set_free (char **elements, size_t nummemebers)

  Frees the returned array from kredis_set_get()
- bool kredis_hash_set (kredis_t *k, const char *key, const char *field, const char *value)

  Native Redis HSET Command.
- bool kredis_hash_exists (kredis_t *k, const char *key, const char *field)

  Native Redis HEXISTS Command.
- char * kredis_hash_get (kredis_t *k, const char *key, const char *field)

  Native Redis HGET Command.

5.14.1 Detailed Description

Header file for kredis.c.

5.14.2 Function Documentation

5.14.2.1 kredis_connect()

```
kredis_t* kredis_connect (
            const char * ip,
            int port,
            bool is_unix_socket,
            struct timeval timeout,
            const char * auth )
```

Connects to a redis server.

Parameters

in	*ip*	The IP Address of the server or the path to a unix socket. If NULL set by default to 127.0.0.1.
in	*port*	The port of the redis server. Note: The default redis port is 6379.
in	*is_unix_socket*	Whether ip is a path to a unix socket. port is ignored if set.
in	*timeout*	Sets the connection timeout.
in	*auth*	The redis server password specified in the server config by requirepass. Ignored if equal to NULL.

Returns

On success a kredis context is returned. Otherwise NULL is returned and errno might be set.

The returned kredis context must be freed using kredis_free().

Note

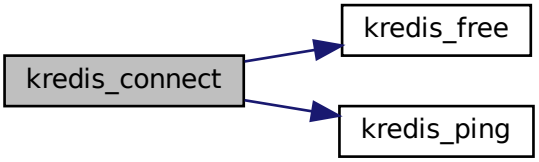
Only the old authentication method prior to Redis 6.0 is supported.

Definition at line 81 of file kredis.c.

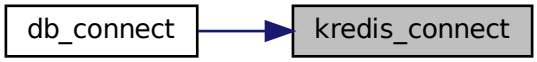
References kredis_free(), kredis_ping(), and kredis_s::rc.

Referenced by db_connect().

Here is the call graph for this function:



Here is the caller graph for this function:



5.14.2.2 kredis_del()

```
bool kredis_del (
            kredis_t * k,
            const char * key )
```

Native Redis DEL Command.

Removes the specified keys. A key is ignored if it does not exist.

See also

https://redis.io/commands/del

Parameters

in,out	*k*	the kredis context
in	*key*	the key

Returns

whether the command was successful.

Return values

false	failed
true	success

Note

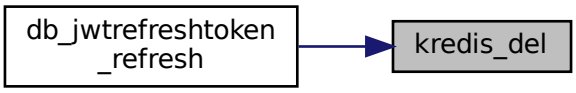
Check for errors using kredis_error() after calling this function.

Definition at line 383 of file kredis.c.

References KREDIS_PREFIX, and kredis_s::rc.

Referenced by db_jwtrefreshtoken_refresh().

Here is the caller graph for this function:



5.14.2.3 kredis_error()

```
int kredis_error (
            kredis_t * k )
```

Gets the current error code as an integer.

In case an error actually happend, you can use kredis_error_string() to get the error message as a string.

Parameters

in	*k*	the kredis context

Returns

the error code as an integer

Return values

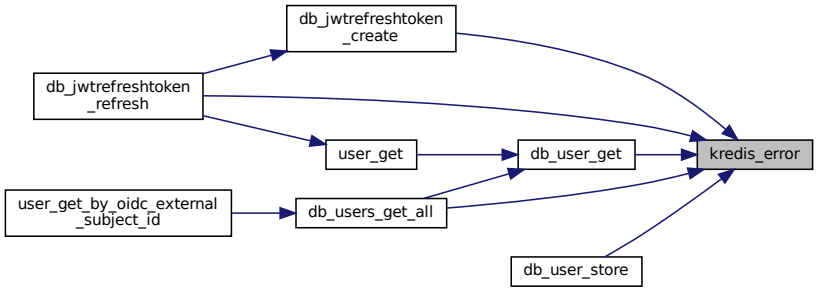
0	indicates no error

Definition at line 192 of file kredis.c.

References kredis_s::rc.

Referenced by db_jwtrefreshtoken_create(), db_jwtrefreshtoken_refresh(), db_user_get(), db_user_store(), and db_users_get_all().

Here is the caller graph for this function:



5.14.2.4 kredis_error_string()

```
const char* kredis_error_string (
            kredis_t * k )
```

Gets the current error code as a string.

This function cannot be used for checking whether an error actually occurred, use kredis_error() instead.

An empty string is returned if no error happend.

Parameters

in	*k*	the kredis context

Returns

the error message

Return values

NULL	invalid argument or invalid redisContext

Note

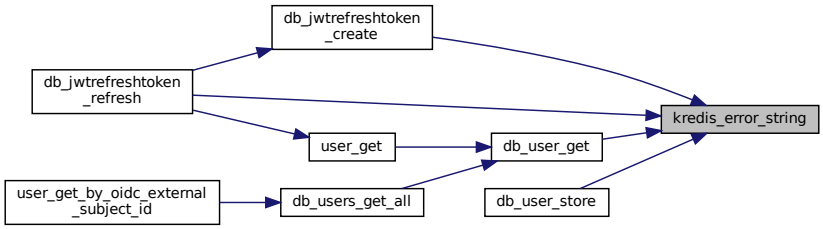
The returned string MUST NOT be freed.

Definition at line 213 of file kredis.c.

References kredis_s::rc.

Referenced by db_jwtrefreshtoken_create(), db_jwtrefreshtoken_refresh(), db_user_get(), and db_user_store().

Here is the caller graph for this function:



5.14.2.5 kredis_free()

```
void kredis_free (
            kredis_t * k )
```

Disconnects and frees a kredis_t object.

Parameters

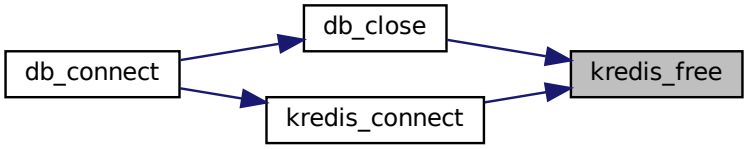
in,out	*k*	the kredis context

Definition at line 57 of file kredis.c.

References kredis_s::rc.

Referenced by db_close(), and kredis_connect().

Here is the caller graph for this function:



5.14.2.6 kredis_get()

```
char* kredis_get (
            kredis_t * k,
            const char * key )
```

Native Redis GET Command.

Get the value of key.

The returned string must be freed using free().

See also

https://redis.io/commands/get

Parameters

in,out	*k*	the kredis context
in	*key*	the key

Returns

the value of key

Return values

NULL	key doesn't exists or isn't a string

Note

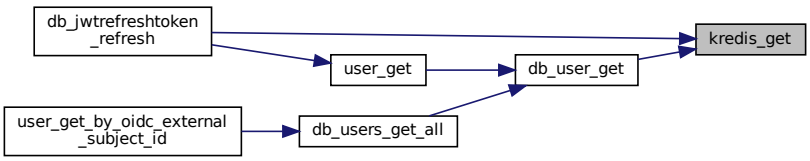
Check for errors using kredis_error() after calling this function.

Definition at line 238 of file kredis.c.

References KREDIS_PREFIX, and kredis_s::rc.

Referenced by db_jwtrefreshtoken_refresh(), and db_user_get().

Here is the caller graph for this function:



5.14.2.7 kredis_hash_exists()

```
bool kredis_hash_exists (
            kredis_t * k,
            const char * key,
            const char * field )
```

Native Redis HEXISTS Command.

Returns if field is an existing field in the hash stored at key.

See also

https://redis.io/commands/hexists

Parameters

in,out	*k*	the kredis context
in	*key*	the key
in	*field*	the field

Returns

whether the field exists in the hash

Return values

true	the hash contains field
false	the hash does not contain field, key does not exist, or other failure

Note

Check for errors using kredis_error() after calling this function.

Definition at line 719 of file kredis.c.

References KREDIS_PREFIX, and kredis_s::rc.

5.14.2.8 kredis_hash_get()

```
char* kredis_hash_get (
            kredis_t * k,
            const char * key,
            const char * field )
```

Native Redis HGET Command.

Returns the value associated with field in the hash stored at key.

The returned string must be freed using free().

See also

https://redis.io/commands/hget

Parameters

in,out	*k*	the kredis context
in	*key*	the key
in	*field*	the field

Returns

the value

Return values

NULL	something failed, errno might be set

Note

Check for errors using kredis_error() after calling this function.

Definition at line 759 of file kredis.c.

References KREDIS_PREFIX, and kredis_s::rc.

5.14.2.9 kredis_hash_set()

```
bool kredis_hash_set (
            kredis_t * k,
            const char * key,
            const char * field,
            const char * value )
```

Native Redis HSET Command.

Sets field in the hash stored at key to value. If key does not exist, a new key holding a hash is created. If field already exists in the hash, it is overwritten.

See also

https://redis.io/commands/hset

Parameters

in,out	*k*	the kredis context
in	*key*	the key
in	*field*	the field
in	*value*	the value

Returns

whether the command was successful.

Return values

false	failed
true	success

Note

Check for errors using kredis_error() after calling this function.

Definition at line 681 of file kredis.c.

References KREDIS_PREFIX, and kredis_s::rc.

5.14.2.10 kredis_ping()

```
bool kredis_ping (
            kredis_t * k )
```

Ping the redis server.

Parameters

in,out	*k*	the kredis context

Returns

whether the ping was successful.

Return values

false	failed
true	success

Note

Check for errors using kredis_error() on failure.

Definition at line 160 of file kredis.c.

References kredis_s::rc.

Referenced by kredis_connect().

Here is the caller graph for this function:

5.14.2.11 kredis_set()

```
bool kredis_set (
            kredis_t * k,
            const char * key,
            const char * value )
```

Native Redis SET Command.

Set key to hold the string value. If key already holds a value, it is overwritten, regardless of its type. Any previous time to live associated with the key is discarded on successful SET operation.

See also

https://redis.io/commands/set

Parameters

in,out	*k*	the kredis context
in	*key*	the key
in	*value*	the value

Returns

whether the command was successful.

Return values

false	failed
true	success

Note

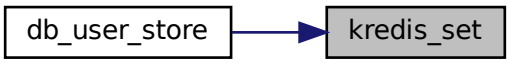
Check for errors using kredis_error() after calling this function.

Definition at line 284 of file kredis.c.

References KREDIS_PREFIX, and kredis_s::rc.

Referenced by db_user_store().

Here is the caller graph for this function:



5.14.2.12 kredis_set_add()

```
bool kredis_set_add (
            kredis_t * k,
            const char * key,
            const char * member )
```

Native Redis SADD Command.

Add the specified member to the set stored at key. Specified members that are already a member of this set are ignored. If key does not exist, a new set is created before adding the specified member.

An error occures when the value stored at key is not a set.

See also

https://redis.io/commands/sadd

Parameters

in,out	*k*	the kredis context
in	*key*	the key
in	*member*	the member to be added

Returns

whether the command was successful.

Return values

false	failed
true	success

Note

Check for errors using kredis_error() after calling this function.

Definition at line 426 of file kredis.c.

References KREDIS_PREFIX, and kredis_s::rc.

Referenced by db_user_store().

Here is the caller graph for this function:

5.14.2.13 kredis_set_free()

```
void kredis_set_free (
            char ** elements,
            size_t nummembers )
```

Frees the returned array from kredis_set_get()

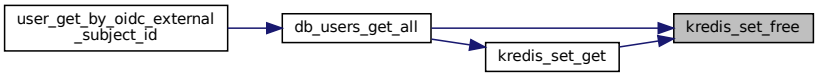
Parameters

in,out	*elements*	the array
in	*nummembers*	the size of the array

Definition at line 650 of file kredis.c.

Referenced by db_users_get_all(), and kredis_set_get().

Here is the caller graph for this function:



5.14.2.14 kredis_set_get()

```
char** kredis_set_get (
            kredis_t * k,
            const char * key,
            size_t * nummembers )
```

Native Redis SMEMBERS Command.

Returns all the members of the set value stored at key.

The returned array must be freed using kredis_set_free().

See also

https://redis.io/commands/smembers

Parameters

in,out	*k*	the kredis context
in	*key*	the key
out	*nummembers*	the number of elements that were read and thus the size of the returned array

Returns

an array of strings

Return values

NULL	something failed, errno might be set

Note

Check for errors using kredis_error() after calling this function.

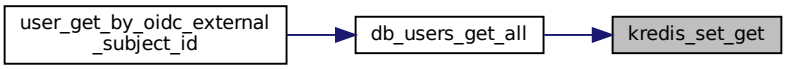
Definition at line 585 of file kredis.c.

References KREDIS_PREFIX, kredis_set_free(), and kredis_s::rc.

Referenced by db_users_get_all().

Here is the call graph for this function:

Here is the caller graph for this function:



5.14.2.15 kredis_set_ismember()

```
bool kredis_set_ismember (
            kredis_t * k,
            const char * key,
            const char * member )
```

Native Redis SISMEMBER Command.

Returns if member is a member of the set stored at key.

See also

https://redis.io/commands/sismember

Parameters

in,out	*k*	the kredis context
in	*key*	the key
in	*member*	the member

Returns

whether member is a member of the set

Return values

true	the element is a member of the set
false	the element is not a member of the set, key does not exist, or other failure

Note

Check for errors using kredis_error() after calling this function.

Definition at line 503 of file kredis.c.

References KREDIS_PREFIX, and kredis_s::rc.

5.14.2.16 kredis_set_num()

```
long long kredis_set_num (
            kredis_t * k,
            const char * key )
```

Native Redis SCARD Command.

Returns the set cardinality (number of elements) of the set stored at key.

See also

https://redis.io/commands/scard

Parameters

in,out	*k*	the kredis context
in	*key*	the key

Returns

the set cardinality

Return values

0	failed or there are no elements in the set

Note

Check for errors using kredis_error() after calling this function.

Definition at line 463 of file kredis.c.

References KREDIS_PREFIX, and kredis_s::rc.

5.14.2.17 kredis_set_rem()

```
bool kredis_set_rem (
            kredis_t * k,
            const char * key,
            const char * member )
```

Native Redis SREM Command.

Remove the specified member from the set stored at key. Specified members that are not a member of this set are ignored. If key does not exist, it is treated as an empty set.

An error occures when the value stored at key is not a set.

See also

https://redis.io/commands/srem

Parameters

in,out	*k*	the kredis context
in	*key*	the key
in	*member*	the member to be removed

Returns

whether the command was successful.

Return values

false	failed
true	success

Note

Check for errors using kredis_error() after calling this function.

Definition at line 546 of file kredis.c.

References KREDIS_PREFIX, and kredis_s::rc.

5.14.2.18 kredis_set_with_ex()

```
bool kredis_set_with_ex (
            kredis_t * k,
            const char * key,
            const char * value,
            long ex )
```

Native Redis SET Command with EX set.

Set key to hold the string value. If key already holds a value, it is overwritten, regardless of its type. Any previous time to live associated with the key is discarded on successful SET operation.

See also

https://redis.io/commands/set

Parameters

in,out	*k*	the kredis context
in	*key*	the key
in	*value*	the value
in	*ex*	expire time, in seconds

Returns

whether the command was successful.

Return values

false	failed
true	success

Note

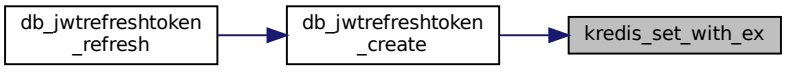
Check for errors using kredis_error() after calling this function.

Definition at line 333 of file kredis.c.

References KREDIS_PREFIX, and kredis_s::rc.

Referenced by db_jwtrefreshtoken_create().

Here is the caller graph for this function:

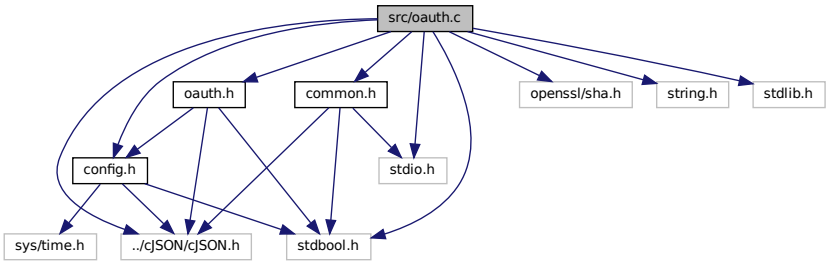


5.15 src/oauth.c File Reference

Useful code for OAuth Authentication.

```
#include "oauth.h"
#include "common.h"
#include "config.h"
#include "../cJSON/cJSON.h"
#include <openssl/sha.h>
#include <string.h>
#include <stdlib.h>
#include <stdio.h>
#include <stdbool.h>
```

Include dependency graph for oauth.c:



Functions

- bool oauth_verify_state_encoding (const char *state)

  Tests whether a state is encoded correctly.
- char * oauth_gen_state (void)

  Generates a STATE for OAuth.
- char * oauth_gen_code_verifier (void)

  Generates a CODE_VERIFIER for OAuth.
- char * oauth_gen_code_challenge (const char *code_verifier)

  Generates a CODE_CHALLENGE for OAuth.
- char * oauth_gen_url (char *errmsg, const struct config_auth_s *authconfig, const char *state)

  Generates an OAuth authorization Url.

5.15.1 Detailed Description

Useful code for OAuth Authentication.

5.15.2 Function Documentation

5.15.2.1 oauth_gen_code_challenge()

```
char* oauth_gen_code_challenge (
            const char * code_verifier )
```

Generates a CODE_CHALLENGE for OAuth.

This is a URL-safe base64-encoded (without padding) string of the SHA256 hash of the CODE_VERIFIER.

The returned string is already percent encoded.

See also

https://docs.gitlab.com/ee/api/oauth2.html#authorization-code-with-proof-key-for-co

Parameters

in	*code_verifier*	the CODE_VERIFIER needed for generating the CODE_CHALLENGE

Returns

the CODE_CHALLENGE represented by a string

Return values

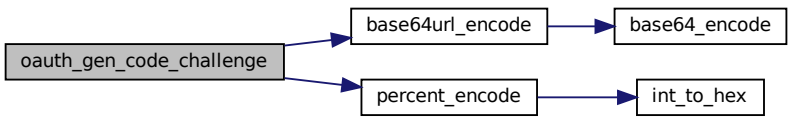
NULL	an error occurred and errno might be set

The returned string must be freed using free().

Definition at line 104 of file oauth.c.

References base64url_encode(), and percent_encode().

Here is the call graph for this function:



5.15.2.2 oauth_gen_code_verifier()

```
char* oauth_gen_code_verifier (
            void  )
```

Generates a CODE_VERIFIER for OAuth.

This is basically just two consecutive UUIDv4s.

See also

https://docs.gitlab.com/ee/api/oauth2.html#authorization-code-with-proof-key-for-co

Returns

a new CODE_VERIFIER represented by a string

Return values

NULL	an error occurred and errno might be set

The returned string must be freed using free().

Definition at line 70 of file oauth.c.

References uuid_v4_gen(), and UUIDLEN.

Here is the call graph for this function:

5.15.2.3 oauth_gen_state()

```
char* oauth_gen_state (
            void  )
```

Generates a STATE for OAuth.

This is basically just a UUIDv4.

See also

https://docs.gitlab.com/ee/api/oauth2.html#authorization-code-with-proof-key-for-co

Returns

a new STATE represented by a string

Return values

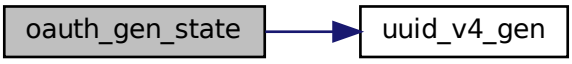
NULL	an error occurred and errno might be set

The returned string must be freed using free().

Definition at line 46 of file oauth.c.

References uuid_v4_gen(), and UUIDLEN.

Here is the call graph for this function:



5.15.2.4 oauth_gen_url()

```
char* oauth_gen_url (
            char * errmsg,
            const struct config_auth_s * authconfig,
            const char * state )
```

Generates an OAuth authorization Url.

An example authconfig can be found in config.example.json["auth"].

See also

https://docs.gitlab.com/ee/api/oauth2.html#authorization-code-flow

Parameters

out	*errmsg*	buffer for error messages. size: ERRMSGMAXSIZE
in	*authconfig*	authentication settings
in	*state*	see oauth_gen_state()

Returns

A fully usable OAuth authorization Url.

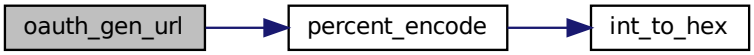
Return values

NULL	an error occurred. errno might be set and errmsg contains an error message.

Definition at line 134 of file oauth.c.

References config_auth_s::app_id, config_auth_s::oauth_authorize_url, percent_encode(), config_auth_s::redirect_uri, config_auth_s::scope, and config_auth_s::type.

Here is the call graph for this function:



5.15.2.5 oauth_verify_state_encoding()

```
bool oauth_verify_state_encoding (
            const char * state )
```

Tests whether a state is encoded correctly.

Parameters

in	*state*	see oauth_gen_state()

Returns

whether the state is encoded correctly

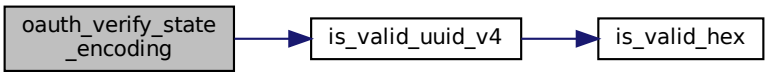
Return values

true	valid encoding; safe to use
false	invalid encoding; dangerous string!

Definition at line 30 of file oauth.c.

References is_valid_uuid_v4().

Here is the call graph for this function:

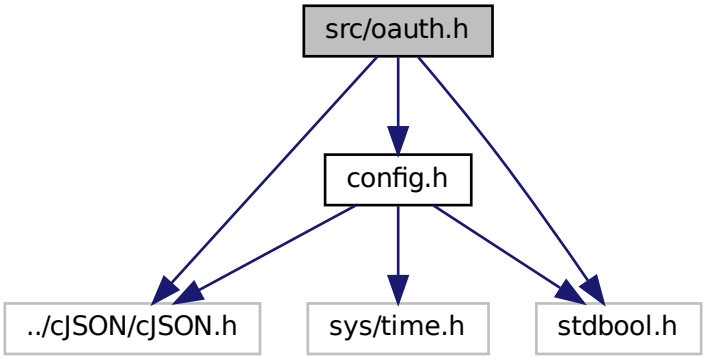


5.16 src/oauth.h File Reference

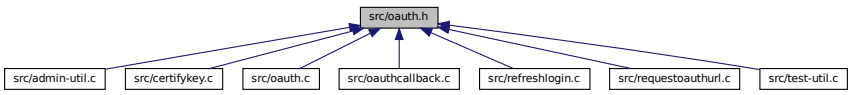
Header file for oauth.c.

```
#include "config.h"
#include "../cJSON/cJSON.h"
#include <stdbool.h>
```

Include dependency graph for oauth.h:



This graph shows which files directly or indirectly include this file:



Functions

- bool oauth_verify_state_encoding (const char *state)

Tests whether a state is encoded correctly.

- char * oauth_gen_state (void)

  Generates a STATE for OAuth.
- char * oauth_gen_code_verifier (void)

  Generates a CODE_VERIFIER for OAuth.
- char * oauth_gen_code_challenge (const char *code_verifier)

  Generates a CODE_CHALLENGE for OAuth.
- char * oauth_gen_url (char *errmsg, const struct config_auth_s *authconfig, const char *state)

  Generates an OAuth authorization Url.

5.16.1 Detailed Description

Header file for oauth.c.

5.16.2 Function Documentation

5.16.2.1 oauth_gen_code_challenge()

```
char* oauth_gen_code_challenge (
            const char * code_verifier )
```

Generates a CODE_CHALLENGE for OAuth.

This is a URL-safe base64-encoded (without padding) string of the SHA256 hash of the CODE_VERIFIER.

The returned string is already percent encoded.

See also

https://docs.gitlab.com/ee/api/oauth2.html#authorization-code-with-proof-key-for-co

Parameters

in	*code_verifier*	the CODE_VERIFIER needed for generating the CODE_CHALLENGE

Returns

the CODE_CHALLENGE represented by a string

Return values

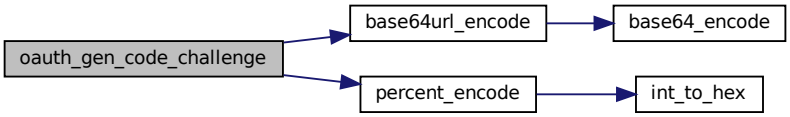
NULL	an error occurred and errno might be set

The returned string must be freed using free().

Definition at line 104 of file oauth.c.

References base64url_encode(), and percent_encode().

Here is the call graph for this function:



5.16.2.2 oauth_gen_code_verifier()

```
char* oauth_gen_code_verifier (
            void  )
```

Generates a CODE_VERIFIER for OAuth.

This is basically just two consecutive UUIDv4s.

See also

https://docs.gitlab.com/ee/api/oauth2.html#authorization-code-with-proof-key-for-co

Returns

a new CODE_VERIFIER represented by a string

Return values

NULL	an error occurred and errno might be set

The returned string must be freed using free().

Definition at line 70 of file oauth.c.

References uuid_v4_gen(), and UUIDLEN.

Here is the call graph for this function:

5.16.2.3 oauth_gen_state()

```
char* oauth_gen_state (
            void  )
```

Generates a STATE for OAuth.

This is basically just a UUIDv4.

See also

https://docs.gitlab.com/ee/api/oauth2.html#authorization-code-with-proof-key-for-co

Returns

a new STATE represented by a string

Return values

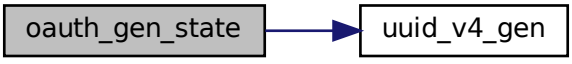
NULL	an error occurred and errno might be set

The returned string must be freed using free().

Definition at line 46 of file oauth.c.

References uuid_v4_gen(), and UUIDLEN.

Here is the call graph for this function:



5.16.2.4 oauth_gen_url()

```
char* oauth_gen_url (
            char * errmsg,
            const struct config_auth_s * authconfig,
            const char * state )
```

Generates an OAuth authorization Url.

An example authconfig can be found in config.example.json["auth"].

See also

https://docs.gitlab.com/ee/api/oauth2.html#authorization-code-flow

Parameters

out	*errmsg*	buffer for error messages. size: ERRMSGMAXSIZE
in	*authconfig*	authentication settings
in	*state*	see oauth_gen_state()

Returns

A fully usable OAuth authorization Url.

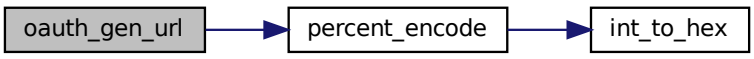
Return values

NULL	an error occurred. errno might be set and errmsg contains an error message.

Definition at line 134 of file oauth.c.

References config_auth_s::app_id, config_auth_s::oauth_authorize_url, percent_encode(), config_auth_s::redirect_uri, config_auth_s::scope, and config_auth_s::type.

Here is the call graph for this function:



5.16.2.5 oauth_verify_state_encoding()

```
bool oauth_verify_state_encoding (
            const char * state )
```

Tests whether a state is encoded correctly.

Parameters

in	*state*	see oauth_gen_state()

Returns

whether the state is encoded correctly

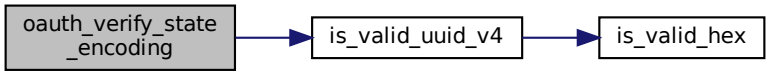
Return values

true	valid encoding; safe to use
false	invalid encoding; dangerous string!

Definition at line 30 of file oauth.c.

References is_valid_uuid_v4().

Here is the call graph for this function:

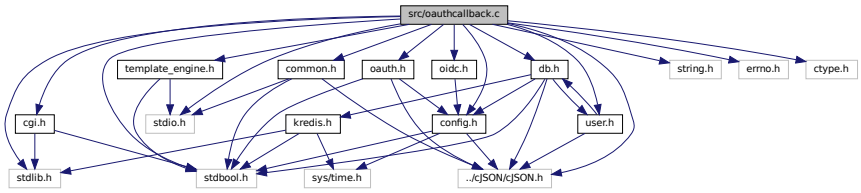


5.17 src/oauthcallback.c File Reference

oauth callback receiver. handles endpoint at /cgi/oauthcallback

```
#include "common.h"
#include "cgi.h"
#include "db.h"
#include "oauth.h"
#include "config.h"
#include "user.h"
#include "oidc.h"
#include "template_engine.h"
#include "../cJSON/cJSON.h"
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include <string.h>
#include <errno.h>
```

`#include <ctype.h>`
Include dependency graph for oauthcallback.c:



Functions

- int main (void)

  main function for oauthcallback

5.17.1 Detailed Description

oauth callback receiver. handles endpoint at /cgi/oauthcallback

Request: Expects a GET Request. The QUERY_STRING should contain:

- `code` the oauth code
- `state` the oauth state

Response: Content-Type: `text/html` The Body should contain the rendered version of access_granted.html or access_denied.html.

5.17.2 Function Documentation

5.17.2.1 main()

```
int main (
            void  )
```

main function for oauthcallback

Returns

the exit code of this program.

Return values

0	success
!0	failure

Definition at line 44 of file oauthcallback.c.

References cgi_testenv(), config_s::db, DB_JWTREFRESHTOKEN_BUFSIZE, and ERRMSGMAXSIZE.

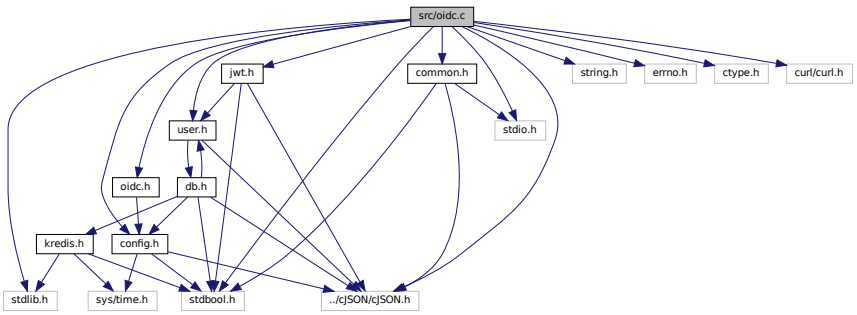
Here is the call graph for this function:

5.18 src/oidc.c File Reference

Open ID Connect implementation.

```
#include "oidc.h"
#include "user.h"
#include "common.h"
#include "config.h"
#include "jwt.h"
#include "../cJSON/cJSON.h"
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include <string.h>
#include <errno.h>
#include <ctype.h>
#include <curl/curl.h>
```

Include dependency graph for oidc.c:



Data Structures

- struct oidc_memory_s

  internal struct used for receiving curl callback data

Functions

- char * oidc_get_external_subject_id_from_identity_provider (char *errmsg, const struct config_auth_s *authconfig, const char *code)

  Get external subject id from identity provider.

5.18.1 Detailed Description

Open ID Connect implementation.

See also

oauth.c

oauthcallback.c

https://goteleport.com/blog/how-oidc-authentication-works/

5.18.2 Function Documentation

5.18.2.1 oidc_get_external_subject_id_from_identity_provider()

```
char* oidc_get_external_subject_id_from_identity_provider (
            char * errmsg,
            const struct config_auth_s * authconfig,
            const char * code )
```

Get external subject id from identity provider.

This function will use the oauth code received by the oauth callback endpoint to get the OpenID Connect id_token from the identity provider specified in the config.

Parameters

out	*errmsg*	buffer for error messages. size: ERRMSGMAXSIZE
in	*authconfig*	authentication settings
in	*code*	the oauth code oauthcallback received

Returns

the oidc external subject id of the user that just authenticated at the identity provider

Return values

NULL	an error occurred. errno might be set and errmsg contains an error message

The returned string must be freed using free().

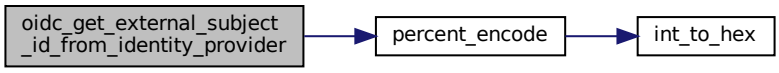
See also

user_get_by_oidc_external_subject_id()

Definition at line 74 of file oidc.c.

References config_auth_s::app_id, config_auth_s::app_secret, oidc_memory_s::memory, config_auth_s::oauth_↩ token_url, percent_encode(), config_auth_s::redirect_uri, oidc_memory_s::size, and config_auth_s::type.

Here is the call graph for this function:

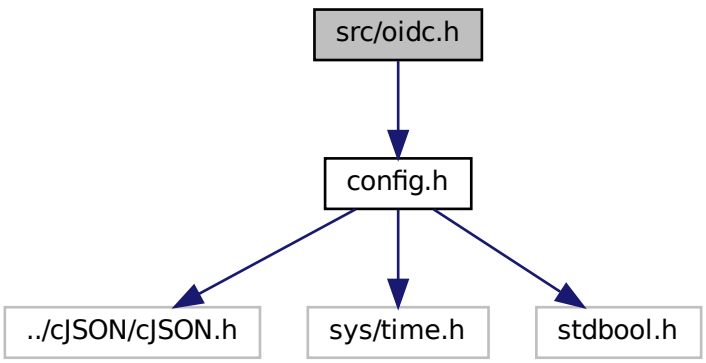


5.19 src/oidc.h File Reference

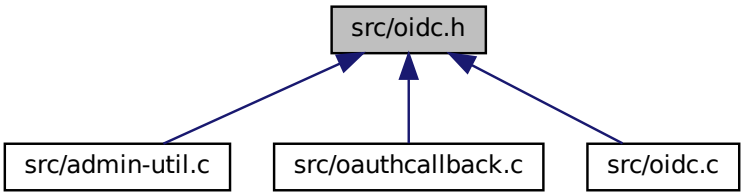
Header file for oidc.c.

```
#include "config.h"
```

Include dependency graph for oidc.h:



This graph shows which files directly or indirectly include this file:



Functions

- char * oidc_get_external_subject_id_from_identity_provider (char *errmsg, const struct config_auth_s *authconfig, const char *code)

  Get external subject id from identity provider.

5.19.1 Detailed Description

Header file for oidc.c.

5.19.2 Function Documentation

5.19.2.1 oidc_get_external_subject_id_from_identity_provider()

```
char* oidc_get_external_subject_id_from_identity_provider (
            char * errmsg,
            const struct config_auth_s * authconfig,
            const char * code )
```

Get external subject id from identity provider.

This function will use the oauth code received by the oauth callback endpoint to get the OpenID Connect id_token from the identity provider specified in the config.

Parameters

out	*errmsg*	buffer for error messages. size: ERRMSGMAXSIZE
in	*authconfig*	authentication settings
in	*code*	the oauth code oauthcallback received

Returns

the oidc external subject id of the user that just authenticated at the identity provider

Return values

NULL	an error occurred. errno might be set and errmsg contains an error message

The returned string must be freed using free().

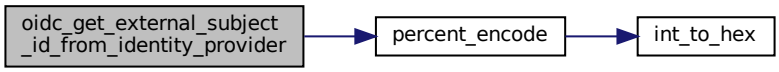
See also

user_get_by_oidc_external_subject_id()

Definition at line 74 of file oidc.c.

References config_auth_s::app_id, config_auth_s::app_secret, oidc_memory_s::memory, config_auth_s::oauth_↩ token_url, percent_encode(), config_auth_s::redirect_uri, oidc_memory_s::size, and config_auth_s::type.

Here is the call graph for this function:

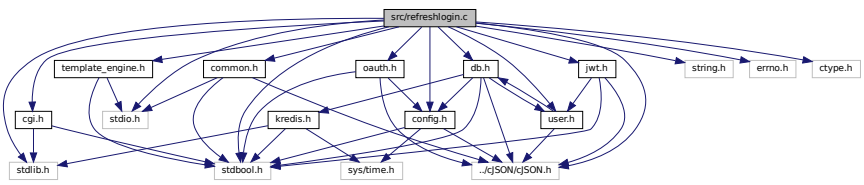


5.20 src/refreshlogin.c File Reference

utility for refreshing the jwtrefreshtoken and getting a JWT. handles endpoint at /cgi/refreshlogin

```
#include "common.h"
#include "cgi.h"
#include "db.h"
#include "jwt.h"
#include "oauth.h"
#include "config.h"
#include "user.h"
#include "template_engine.h"
#include "../cJSON/cJSON.h"
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include <string.h>
#include <errno.h>
#include <ctype.h>
```

Include dependency graph for refreshlogin.c:



Functions

- int main (void)

  main function for refreshlogin

5.20.1 Detailed Description

utility for refreshing the jwtrefreshtoken and getting a JWT. handles endpoint at /cgi/refreshlogin

Request: Expects a POST Request with Content-Type set to `application/x-www-form-urlencoded`. The Request Header must contain a valid jwtrefreshtoken cookie. The User must be 'active'. The Body should be empty. The QUERY_STRING may contain a `as` field specifying the Content-Type of the response. (Supported are `json`,`html` and `plain`. Default: `json`)

Response as json (if successful): Content-Type: `application/json` Set-Cookie: `jwtrefreshtoken` The Body should contain a JSON Object with a `jwt` string field and an `expires_in` number field.

Response as html (if successful): Content-Type: `text/html` Set-Cookie: `jwtrefreshtoken` The Body should contain the rendered version of refreshlogin.html.

Response as plain (if successful): Content-Type: `text/plain` Set-Cookie: `jwtrefreshtoken` The Body should contain the JWT.

5.20.2 Function Documentation

5.20.2.1 main()

```
int main (
            void  )
```

main function for refreshlogin

Returns

the exit code of this program.

Return values

0	success
!0	failure

Definition at line 58 of file refreshlogin.c.

References cgi_testenv(), config_s::db, DB_JWTREFRESHTOKEN_BUFSIZE, and ERRMSGMAXSIZE.

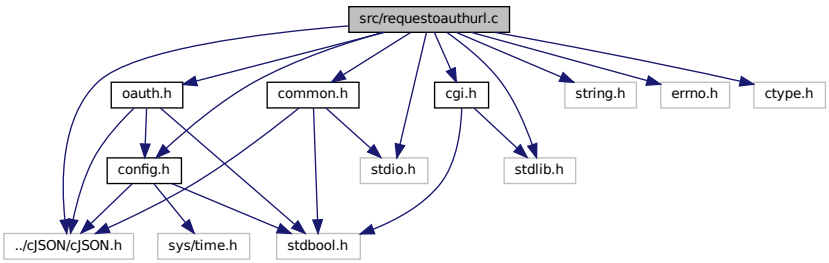
Here is the call graph for this function:

5.21 src/requestoauthurl.c File Reference

oauth url generator. handles endpoint at /cgi/requestoauthurl

```
#include "common.h"
#include "cgi.h"
#include "oauth.h"
#include "config.h"
#include "../cJSON/cJSON.h"
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <errno.h>
#include <ctype.h>
```

Include dependency graph for requestoauthurl.c:



Functions

- int main (void)

  main function for requestoauthurl

5.21.1 Detailed Description

oauth url generator. handles endpoint at /cgi/requestoauthurl

Request: Expects a GET Request. The QUERY_STRING may contain a `as` field specifying the Content-Type of the response. (Supported are `json`,`redirect` and `plain`. Default: `json`)

Response as json (if successful): Content-Type: `application/json` The Body should contain a JSON Object with an `oauthurl` string field.

Response as plain (if successful): Content-Type: `text/plain` The Body should contain `oauthurl`.

Response as redirect (if successful): Content-Type: `text/html` Status: 302 Location: `oauthurl`

5.21.2 Function Documentation

5.21.2.1 main()

```
int main (
            void  )
```

main function for requestoauthurl

Returns

the exit code of this program.

Return values

0	success
!0	failure

Definition at line 47 of file requestoauthurl.c.

References cgi_testenv(), and ERRMSGMAXSIZE.

Here is the call graph for this function:

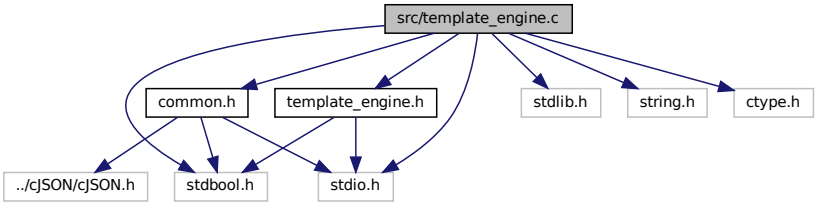
5.22 src/template_engine.c File Reference

Template rendering code.

```
#include "template_engine.h"
#include "common.h"
#include <stdbool.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
```

```
#include <ctype.h>
```

Include dependency graph for template_engine.c:



Functions

- char * tmplt_render (char *errmsg, const char *tmplt, const struct tmplt_kvp_s *data)

  Renders a template to a string.
- char * tmplt_render_from_file (char *errmsg, const char *tmplt_file_path, const struct tmplt_kvp_s *data)

  Renders a template to a string.

5.22.1 Detailed Description

Template rendering code.

5.22.1.1 TEMPLATING LANGUAGE

The data model consits of key-value-pairs. A key in a template is marked by enclosing it in pairs of percent-signs. For example: "Hello %%name%%. I like you." Note: A template is invalid if a key doesn't exist.

5.22.2 Function Documentation

5.22.2.1 tmplt_render()

```
char* tmplt_render (
            char * errmsg,
            const char * tmplt,
            const struct tmplt_kvp_s * data )
```

Renders a template to a string.

This will return a copy of tmplt where all instances of `%KEY%%` are replaced with the associated value in data.

See also

TEMPLATING LANGUAGE // TODO reference section

A value marked as unsafe will be percent encoded using percent_encode().

Parameters

out	*errmsg*	buffer for error messages. size: ERRMSGMAXSIZE
in	*tmplt*	the template as a string
in	*data*	a null terminated key-value-pair array

Returns

the rendered document as a string

Return values

NULL	an error occurred and an error message has been written to errmsg. errno might be set.

The returned string must be freed using free().

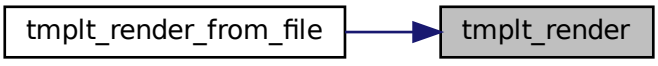
Definition at line 66 of file template_engine.c.

References tmplt_kvp_s::key, str_append(), TEMPLATE_KEY_CLOSE, and TEMPLATE_KEY_OPEN.

Referenced by tmplt_render_from_file().

Here is the call graph for this function:

Here is the caller graph for this function:



5.22.2.2 tmplt_render_from_file()

```
char* tmplt_render_from_file (
            char * errmsg,
            const char * tmplt_file_path,
            const struct tmplt_kvp_s * data )
```

Renders a template to a string.

Wrapper for tmplt_render().

See also

tmplt_render()

Parameters

out	*errmsg*	buffer for error messages. size: ERRMSGMAXSIZE
in	*tmplt_file_path*	path to the template file
in	*data*	a null terminated key-value-pair array

Returns

the rendered document as a string

Return values

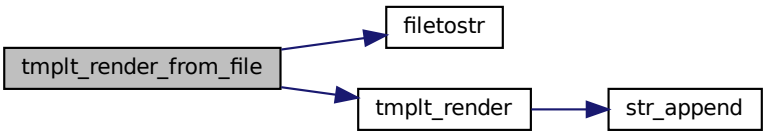
NULL	an error occurred and an error message has been written to errmsg. errno might be set.

The returned string must be freed using free().

Definition at line 184 of file template_engine.c.

References filetostr(), and tmplt_render().

Here is the call graph for this function:

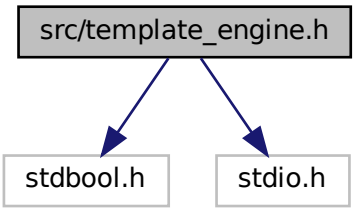


5.23 src/template_engine.h File Reference

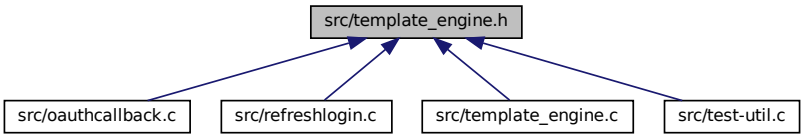
Header file for template_engine.c.

```
#include <stdbool.h>
#include <stdio.h>
```

Include dependency graph for template_engine.h:



This graph shows which files directly or indirectly include this file:



Data Structures

- struct tmplt_kvp_s

  Template Key-Value-Pair Struct.

Macros

- #define TEMPLATES_DIR "../../frontend/templates"

  Path to the directory containing the templates.
- #define TEMPLATE_KEY_OPEN "%%"

  Indicator for the beginning of a key.
- #define TEMPLATE_KEY_CLOSE "%%"

  Indicator for the ending of a key.

Functions

- char * tmplt_render (char *errmsg, const char *tmplt, const struct tmplt_kvp_s *data)

  Renders a template to a string.
- char * tmplt_render_from_file (char *errmsg, const char *tmplt_file_path, const struct tmplt_kvp_s *data)

  Renders a template to a string.

5.23.1 Detailed Description

Header file for template_engine.c.

5.23.2 Function Documentation

5.23.2.1 tmplt_render()

```
char* tmplt_render (
            char * errmsg,
            const char * tmplt,
            const struct tmplt_kvp_s * data )
```

Renders a template to a string.

This will return a copy of tmplt where all instances of `%KEY%%` are replaced with the associated value in data.

See also

TEMPLATING LANGUAGE // TODO reference section

A value marked as unsafe will be percent encoded using percent_encode().

Parameters

out	*errmsg*	buffer for error messages. size: ERRMSGMAXSIZE
in	*tmplt*	the template as a string
in	*data*	a null terminated key-value-pair array

Returns

the rendered document as a string

Return values

NULL	an error occurred and an error message has been written to errmsg. errno might be set.

The returned string must be freed using free().

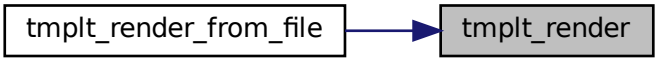
Definition at line 66 of file template_engine.c.

References tmplt_kvp_s::key, str_append(), TEMPLATE_KEY_CLOSE, and TEMPLATE_KEY_OPEN.

Referenced by tmplt_render_from_file().

Here is the call graph for this function:

Here is the caller graph for this function:



5.23.2.2 tmplt_render_from_file()

```
char* tmplt_render_from_file (
            char * errmsg,
            const char * tmplt_file_path,
            const struct tmplt_kvp_s * data )
```

Renders a template to a string.

Wrapper for tmplt_render().

See also

tmplt_render()

Parameters

out	*errmsg*	buffer for error messages. size: ERRMSGMAXSIZE
in	*tmplt_file_path*	path to the template file
in	*data*	a null terminated key-value-pair array

Returns

the rendered document as a string

Return values

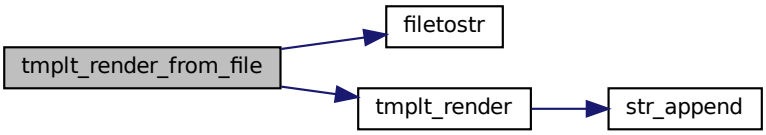
NULL	an error occurred and an error message has been written to errmsg. errno might be set.

The returned string must be freed using free().

Definition at line 184 of file template_engine.c.

References filetostr(), and tmplt_render().

Here is the call graph for this function:

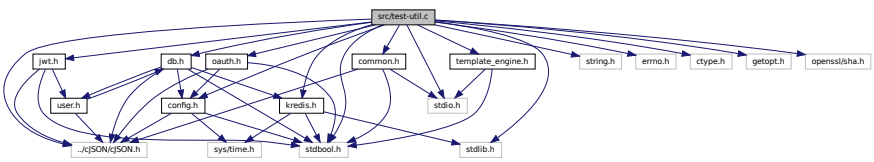


5.24 src/test-util.c File Reference

Tool for testing PARTs of this program.

```
#include "common.h"
#include "db.h"
#include "oauth.h"
#include "kredis.h"
#include "jwt.h"
#include "config.h"
#include "template_engine.h"
#include "../cJSON/cJSON.h"
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include <string.h>
#include <errno.h>
#include <ctype.h>
#include <getopt.h>
#include <openssl/sha.h>
```

Include dependency graph for test-util.c:



Data Structures

- struct testable_s

  testcase struct
- struct encoder_testcase_s

  struct used for testing encoders

Macros

- #define MAXFILEPATHLEN 256

  the maximum length of a file path

Functions

- int main (int argc, char **argv, char **envp)

  main function for test-util

5.24.1 Detailed Description

Tool for testing PARTs of this program.

5.24.2 Function Documentation

5.24.2.1 main()

```
int main (
            int argc,
            char ** argv,
            char ** envp )
```

main function for test-util

Parameters

in	*argc*	number of arguments from the command line
in	*argv*	arguments from the command line
in	*envp*	environment variables

Returns

the exit code of this program.

Return values

0	success
!0	failure

Definition at line 980 of file test-util.c.

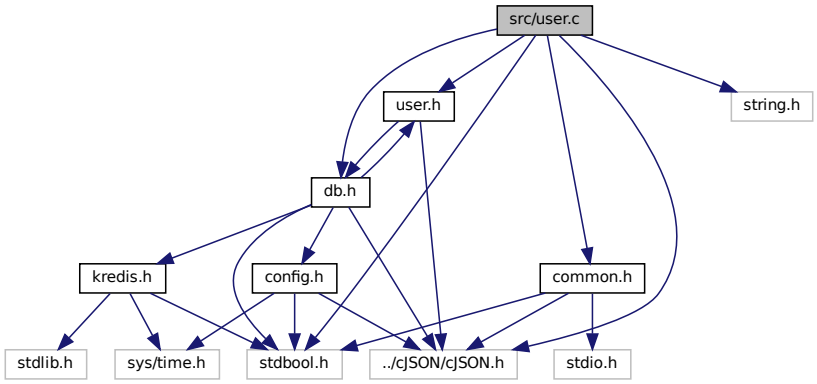
References ERRMSGMAXSIZE, and MAXFILEPATHLEN.

5.25 src/user.c File Reference

User related code.

```
#include "user.h"
#include "common.h"
#include "db.h"
#include "../cJSON/cJSON.h"
#include <string.h>
#include <stdbool.h>
```

Include dependency graph for user.c:



Functions

- user_t * user_get_by_oidc_external_subject_id (char *errmsg, db_t *db, const char *sub)

  Retrieves a user from the database via their oidc external subject id.
- user_t * user_get (char *errmsg, db_t *db, const char *userid)

  Retrieves a user from the database via their userid.
- user_t * user_parse (char *errmsg, const cJSON *userjson)

  Parses a user from json.
- void user_free (user_t *user)

  Frees a user.

5.25.1 Detailed Description

User related code.

5.25.2 Function Documentation

5.25.2.1 user_free()

```
void user_free (
            user_t * user )
```

Frees a user.

Parameters

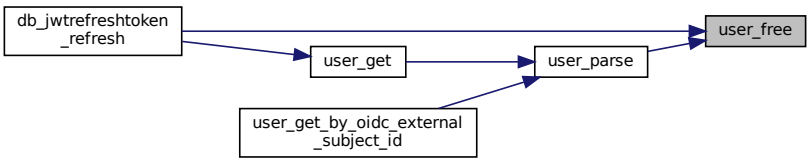
in	*user*	the user

Definition at line 212 of file user.c.

References user_s::__raw, and user_s::principals.

Referenced by db_jwtrefreshtoken_refresh(), and user_parse().

Here is the caller graph for this function:



5.25.2.2 user_get()

```
user_t* user_get (
            char * errmsg,
            db_t * db,
            const char * userid )
```

Retrieves a user from the database via their userid.

See also

db_user_get()

Parameters

out	*errmsg*	buffer for error messages. size: ERRMSGMAXSIZE
in,out	*db*	the database context
in	*userid*	the users userid

Returns

the user

Return values

NULL	an error occurred or user doesn't exist in the database. An error message has been written to errmsg.

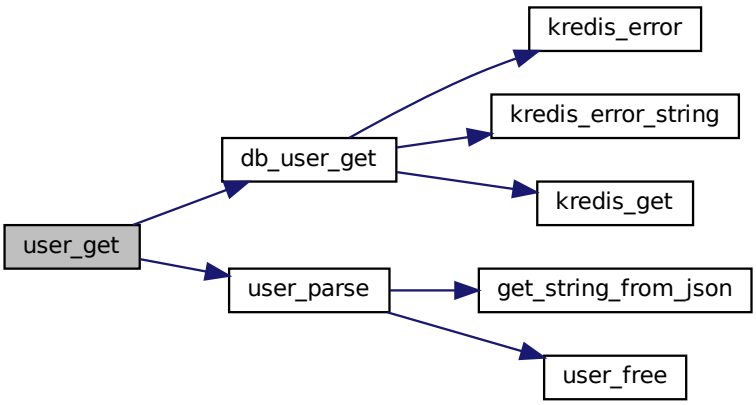
The returned user object must be freed using user_free().

Definition at line 82 of file user.c.

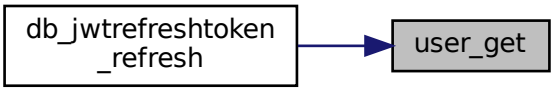
References db_user_get(), testable_s::errmsg, testable_s::res, and user_parse().

Referenced by db_jwtrefreshtoken_refresh().

Here is the call graph for this function:



Here is the caller graph for this function:



5.25.2.3 user_get_by_oidc_external_subject_id()

```
user_t* user_get_by_oidc_external_subject_id (
            char * errmsg,
            db_t * db,
            const char * sub )
```

Retrieves a user from the database via their oidc external subject id.

See also

oidc_get_external_subject_id_from_identity_provider()

Parameters

out	*errmsg*	buffer for error messages. size: ERRMSGMAXSIZE
in,out	*db*	the database context
in	*sub*	the oidc external subject id

Returns

the user

Return values

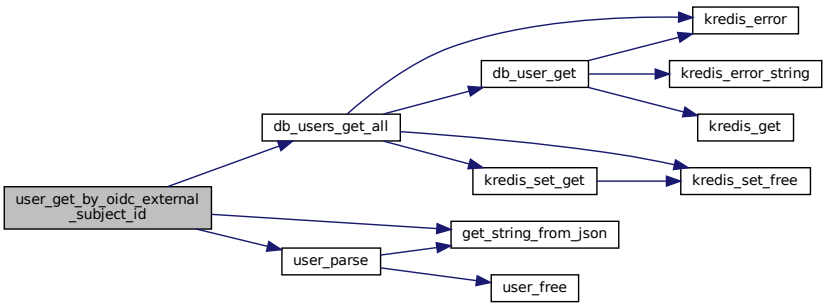
NULL	an error occurred or user doesn't exist in the database. An error message has been written to errmsg.

The returned user object must be freed using user_free().

Definition at line 31 of file user.c.

References db_users_get_all(), testable_s::errmsg, get_string_from_json(), testable_s::res, and user_parse().

Here is the call graph for this function:



5.25.2.4 user_parse()

```
user_t* user_parse (
            char * errmsg,
            const cJSON * userjson )
```

Parses a user from json.

Parameters

out	*errmsg*	buffer for error messages. size: ERRMSGMAXSIZE
in	*userjson*	the user as json

Returns

the parsed user

Return values

NULL	an error occurred and an error message has been written to errmsg

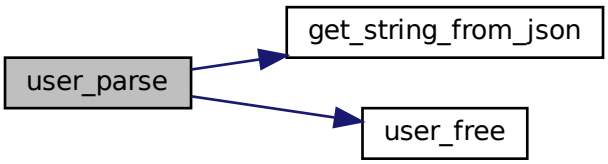
The returned user object must be freed using user_free().

Definition at line 107 of file user.c.

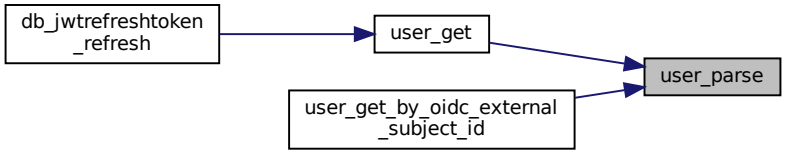
References testable_s::errmsg, get_string_from_json(), testable_s::res, and user_free().

Referenced by user_get(), and user_get_by_oidc_external_subject_id().

Here is the call graph for this function:



Here is the caller graph for this function:

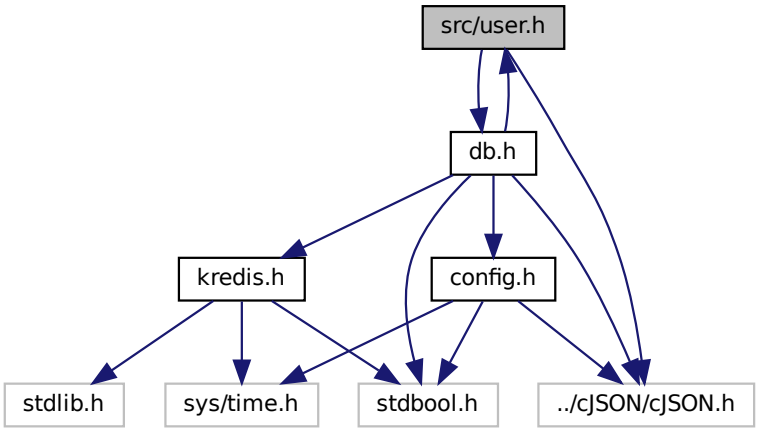


5.26 src/user.h File Reference

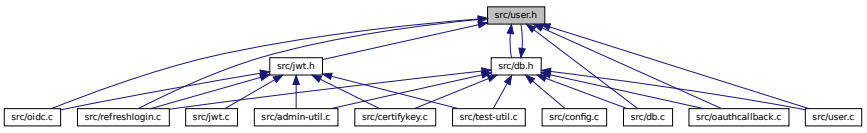
Header file for user.c.

```
#include "db.h"
#include "../cJSON/cJSON.h"
```

Include dependency graph for user.h:



This graph shows which files directly or indirectly include this file:



Data Structures

- struct user_s

  user struct

Typedefs

- typedef struct user_s **user_t**

Functions

- user_t * user_get_by_oidc_external_subject_id (char *errmsg, db_t *db, const char *sub)

  Retrieves a user from the database via their oidc external subject id.
- user_t * user_get (char *errmsg, db_t *db, const char *userid)

  Retrieves a user from the database via their userid.
- user_t * user_parse (char *errmsg, const cJSON *userjson)

  Parses a user from json.
- void user_free (user_t *user)

  Frees a user.

5.26.1 Detailed Description

Header file for user.c.

5.26.2 Function Documentation

5.26.2.1 user_free()

```
void user_free (
            user_t * user )
```

Frees a user.

Parameters

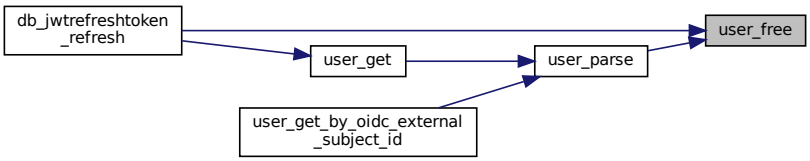
in	*user*	the user

Definition at line 212 of file user.c.

References user_s::__raw, and user_s::principals.

Referenced by db_jwtrefreshtoken_refresh(), and user_parse().

Here is the caller graph for this function:



5.26.2.2 user_get()

```
user_t* user_get (
            char * errmsg,
            db_t * db,
            const char * userid )
```

Retrieves a user from the database via their userid.

See also

db_user_get()

Parameters

out	*errmsg*	buffer for error messages. size: ERRMSGMAXSIZE
in,out	*db*	the database context
in	*userid*	the users userid

Returns

the user

Return values

NULL	an error occurred or user doesn't exist in the database. An error message has been written to errmsg.

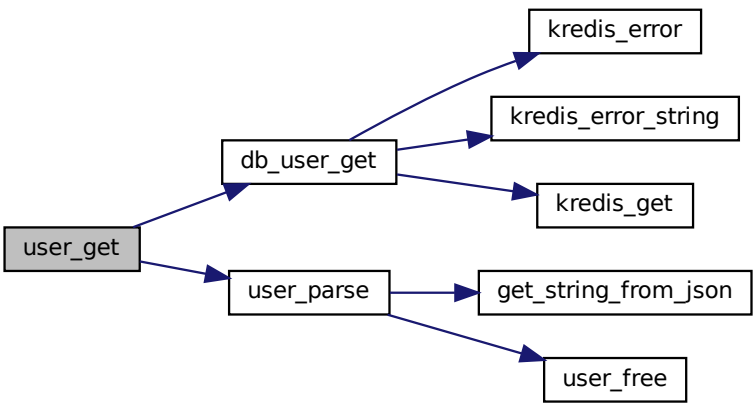
The returned user object must be freed using user_free().

Definition at line 82 of file user.c.

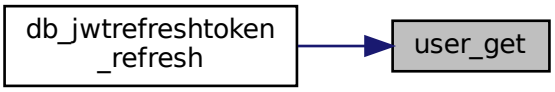
References db_user_get(), testable_s::errmsg, testable_s::res, and user_parse().

Referenced by db_jwtrefreshtoken_refresh().

Here is the call graph for this function:



Here is the caller graph for this function:



5.26.2.3 user_get_by_oidc_external_subject_id()

```
user_t* user_get_by_oidc_external_subject_id (
            char * errmsg,
            db_t * db,
            const char * sub )
```

Retrieves a user from the database via their oidc external subject id.

See also

oidc_get_external_subject_id_from_identity_provider()

Parameters

out	*errmsg*	buffer for error messages. size: ERRMSGMAXSIZE
in,out	*db*	the database context
in	*sub*	the oidc external subject id

Returns

the user

Return values

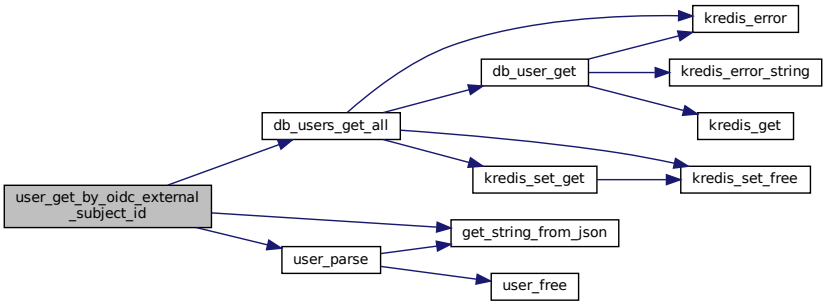
NULL	an error occurred or user doesn't exist in the database. An error message has been written to errmsg.

The returned user object must be freed using user_free().

Definition at line 31 of file user.c.

References db_users_get_all(), testable_s::errmsg, get_string_from_json(), testable_s::res, and user_parse().

Here is the call graph for this function:



5.26.2.4 user_parse()

```
user_t* user_parse (
            char * errmsg,
            const cJSON * userjson )
```

Parses a user from json.

Parameters

out	*errmsg*	buffer for error messages. size: ERRMSGMAXSIZE
in	*userjson*	the user as json

Returns

the parsed user

Return values

NULL	an error occurred and an error message has been written to errmsg

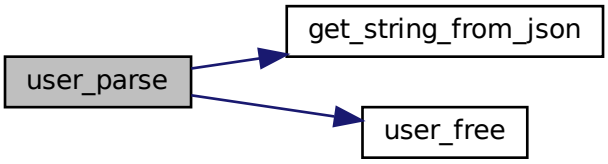
The returned user object must be freed using user_free().

Definition at line 107 of file user.c.

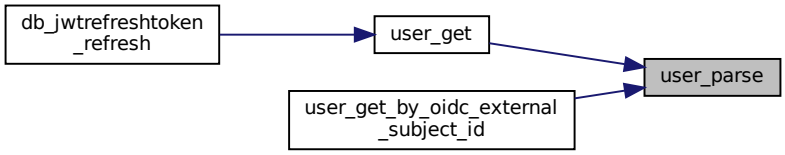
References testable_s::errmsg, get_string_from_json(), testable_s::res, and user_free().

Referenced by user_get(), and user_get_by_oidc_external_subject_id().

Here is the call graph for this function:



Here is the caller graph for this function:



Index