

ADA Platform Adaptation Layer Interfaces

Generated by Doxygen 1.8.11

Contents

1	Class Index	1
1.1	Class List	1
2	File Index	3
2.1	File List	3
3	Class Documentation	5
3.1	al_ada_callback Struct Reference	5
3.1.1	Detailed Description	5
3.1.2	Member Data Documentation	5
3.1.2.1	arg	5
3.1.2.2	func	5
3.1.2.3	next	5
3.1.2.4	pending	6
3.2	al_net_dns_req Struct Reference	6
3.2.1	Detailed Description	6
3.2.2	Member Data Documentation	6
3.2.2.1	addr	6
3.2.2.2	al_priv	6
3.2.2.3	callback	6
3.2.2.4	error	6
3.2.2.5	hostname	6

4	File Documentation	7
4.1	al/al_ada_thread.h File Reference	7
4.1.1	Detailed Description	7
4.1.2	Function Documentation	7
4.1.2.1	al_ada_call(struct al_ada_callback *cb)	7
4.1.2.2	al_ada_callback_init(struct al_ada_callback *cb, void(*func)(void *), void *arg)	8
4.1.2.3	al_ada_kill(void)	8
4.1.2.4	al_ada_main_loop(void)	8
4.1.2.5	al_ada_sync_call(struct al_ada_callback *cb)	8
4.1.2.6	al_ada_timer_cancel(struct timer *timer)	8
4.1.2.7	al_ada_timer_set(struct timer *timer, unsigned long ms)	9
4.1.2.8	al_ada_wakeup(void)	9
4.2	al/al_aes.h File Reference	9
4.2.1	Detailed Description	9
4.2.2	Function Documentation	9
4.2.2.1	al_aes_cbc_decrypt(struct al_aes_ctxt *ctxt, const void *in, void *out, size_t len)	9
4.2.2.2	al_aes_cbc_encrypt(struct al_aes_ctxt *ctxt, const void *in, void *out, size_t len)	10
4.2.2.3	al_aes_cbc_key_set(struct al_aes_ctxt *ctxt, void *key, size_t key_len, void *iv, int decrypt)	10
4.2.2.4	al_aes_ctxt_alloc(void)	10
4.2.2.5	al_aes_ctxt_free(struct al_aes_ctxt *ctxt)	11
4.2.2.6	al_aes_iv_get(struct al_aes_ctxt *ctxt, void *buf, size_t len)	11
4.3	al/al_assert.h File Reference	11
4.3.1	Detailed Description	11
4.3.2	Function Documentation	11
4.3.2.1	al_assert_handle(const char *file, int line)	11
4.4	al/al_cli.h File Reference	12
4.4.1	Detailed Description	12
4.4.2	Function Documentation	12
4.4.2.1	al_cli_print(const char *str)	12

4.4.2.2	al_cli_register(const char *cmd, const char *help, void(*handler)(int argc, char **argv))	12
4.4.2.3	al_cli_set_prompt(const char *pmpt)	13
4.5	al/al_clock.h File Reference	13
4.5.1	Detailed Description	13
4.5.2	Enumeration Type Documentation	13
4.5.2.1	al_clock_src	13
4.5.3	Function Documentation	14
4.5.3.1	al_clock_get(enum al_clock_src *clock_src)	14
4.5.3.2	al_clock_get_total_ms(void)	14
4.5.3.3	al_clock_reset_src(void)	14
4.5.3.4	al_clock_set(u32 timestamp, enum al_clock_src clock_src)	14
4.6	al/al_err.h File Reference	15
4.6.1	Detailed Description	15
4.6.2	Macro Definition Documentation	15
4.6.2.1	AL_ERR_STRINGS	15
4.6.3	Enumeration Type Documentation	16
4.6.3.1	al_err	16
4.6.4	Function Documentation	16
4.6.4.1	al_err_string(enum al_err err)	16
4.7	al/al_hash_sha1.h File Reference	17
4.7.1	Detailed Description	17
4.7.2	Macro Definition Documentation	17
4.7.2.1	AL_HASH_SHA1_SIZE	17
4.7.3	Function Documentation	17
4.7.3.1	al_hash_sha1_add(struct al_hash_sha1_ctxt *ctxt, const void *buf, size_t len)	17
4.7.3.2	al_hash_sha1_ctxt_alloc(void)	17
4.7.3.3	al_hash_sha1_ctxt_free(struct al_hash_sha1_ctxt *ctxt)	18
4.7.3.4	al_hash_sha1_ctxt_init(struct al_hash_sha1_ctxt *ctxt)	18
4.7.3.5	al_hash_sha1_final(struct al_hash_sha1_ctxt *ctxt, void *buf)	18
4.8	al/al_hash_sha256.h File Reference	18

4.8.1	Detailed Description	19
4.8.2	Macro Definition Documentation	19
4.8.2.1	AL_HASH_SHA256_SIZE	19
4.8.3	Function Documentation	19
4.8.3.1	al_hash_sha256_add(struct al_hash_sha256_ctxt *ctxt, const void *buf, size_t len)	19
4.8.3.2	al_hash_sha256_ctxt_alloc(void)	19
4.8.3.3	al_hash_sha256_ctxt_free(struct al_hash_sha256_ctxt *ctxt)	19
4.8.3.4	al_hash_sha256_ctxt_init(struct al_hash_sha256_ctxt *ctxt)	19
4.8.3.5	al_hash_sha256_final(struct al_hash_sha256_ctxt *ctxt, void *buf)	20
4.9	al/al_httpd.h File Reference	20
4.9.1	Detailed Description	21
4.9.2	Enumeration Type Documentation	21
4.9.2.1	al_http_method	21
4.9.3	Function Documentation	21
4.9.3.1	al_httpd_close_conn(struct al_httpd_conn *conn)	21
4.9.3.2	al_httpd_final(void)	21
4.9.3.3	al_httpd_get_method(struct al_httpd_conn *conn)	21
4.9.3.4	al_httpd_get_req_header(struct al_httpd_conn *conn, const char *name)	21
4.9.3.5	al_httpd_get_url(struct al_httpd_conn *conn)	22
4.9.3.6	al_httpd_get_url_arg(struct al_httpd_conn *conn, const char *name)	22
4.9.3.7	al_httpd_get_version(struct al_httpd_conn *conn)	22
4.9.3.8	al_httpd_init(void)	23
4.9.3.9	al_httpd_read(struct al_httpd_conn *conn, char *buf, size_t buf_size)	23
4.9.3.10	al_httpd_reg_url_cb(const char *url, enum al_http_method method, int(*cb)(struct al_httpd_conn *conn))	23
4.9.3.11	al_httpd_response(struct al_httpd_conn *conn, int status, const char *headers)	23
4.9.3.12	al_httpd_start(u16 port)	24
4.9.3.13	al_httpd_stop(void)	24
4.9.3.14	al_httpd_write(struct al_httpd_conn *conn, const char *data, size_t size)	24
4.10	al/al_log.h File Reference	24
4.10.1	Detailed Description	24

4.10.2	Function Documentation	25
4.10.2.1	al_log_get_mod_name(u8 mod_id)	25
4.10.2.2	al_log_print(const char *line)	25
4.11	al/al_net_addr.h File Reference	25
4.11.1	Detailed Description	25
4.11.2	Function Documentation	25
4.11.2.1	al_net_addr_get_ipv4(const struct al_net_addr *addr)	25
4.11.2.2	al_net_addr_set_ipv4(struct al_net_addr *addr, u32 ip)	26
4.12	al/al_net_dns.h File Reference	26
4.12.1	Detailed Description	26
4.12.2	Function Documentation	26
4.12.2.1	al_dns_req_cancel(struct al_net_dns_req *req)	26
4.12.2.2	al_dns_req_ipv4_start(struct al_net_dns_req *req)	27
4.12.2.3	al_net_dns_delete_host(const char *hostname)	27
4.12.2.4	al_net_dns_servers_rotate(void)	27
4.13	al/al_net_if.h File Reference	27
4.13.1	Detailed Description	28
4.13.2	Enumeration Type Documentation	28
4.13.2.1	al_net_if_type	28
4.13.3	Function Documentation	28
4.13.3.1	al_get_net_if(enum al_net_if_type type)	28
4.13.3.2	al_net_if_get_addr(struct al_net_if *net_if)	28
4.13.3.3	al_net_if_get_ipv4(struct al_net_if *net_if)	29
4.13.3.4	al_net_if_get_mac_addr(struct al_net_if *net_if, u8 mac_addr[6])	29
4.13.3.5	al_net_if_get_netmask(struct al_net_if *net_if)	29
4.14	al/al_net_stream.h File Reference	30
4.14.1	Detailed Description	30
4.14.2	Enumeration Type Documentation	30
4.14.2.1	al_net_stream_type	30
4.14.3	Function Documentation	30

4.14.3.1	<code>al_net_stream_close(struct al_net_stream *stream)</code>	30
4.14.3.2	<code>al_net_stream_connect(struct al_net_stream *stream, const char *hostname, struct al_net_addr *host_addr, u16 port, enum al_err(*connected)(void *arg, struct al_net_stream *, enum al_err err))</code>	31
4.14.3.3	<code>al_net_stream_continue_recv(struct al_net_stream *stream)</code>	31
4.14.3.4	<code>al_net_stream_get_tcp_obj(struct al_net_stream *stream)</code>	31
4.14.3.5	<code>al_net_stream_is_established(struct al_net_stream *stream)</code>	32
4.14.3.6	<code>al_net_stream_new(enum al_net_stream_type type)</code>	32
4.14.3.7	<code>al_net_stream_output(struct al_net_stream *stream)</code>	32
4.14.3.8	<code>al_net_stream_recved(struct al_net_stream *stream, size_t len)</code>	32
4.14.3.9	<code>al_net_stream_set_arg(struct al_net_stream *stream, void *arg)</code>	33
4.14.3.10	<code>al_net_stream_set_err_cb(struct al_net_stream *stream, void(*err_cb)(void *arg, enum al_err err))</code>	33
4.14.3.11	<code>al_net_stream_set_recv_cb(struct al_net_stream *stream, enum al_err(*recv_cb)(void *arg, struct al_net_stream *stream, void *data, size_t size))</code>	33
4.14.3.12	<code>al_net_stream_set_sent_cb(struct al_net_stream *stream, void(*sent_cb)(void *arg, struct al_net_stream *stream, size_t len_sent))</code>	33
4.14.3.13	<code>al_net_stream_write(struct al_net_stream *stream, const void *data, size_t len)</code>	34
4.15	<code>al/al_net_udp.h</code> File Reference	34
4.15.1	Detailed Description	34
4.15.2	Function Documentation	35
4.15.2.1	<code>al_net_igmp_joingroup(struct al_net_udp *udp, struct al_net_addr *if_addr, struct al_net_addr *group)</code>	35
4.15.2.2	<code>al_net_igmp_leavegroup(struct al_net_udp *udp, struct al_net_addr *if_addr, struct al_net_addr *group)</code>	35
4.15.2.3	<code>al_net_udp_bind(struct al_net_udp *udp, struct al_net_addr *addr, u16 port)</code>	35
4.15.2.4	<code>al_net_udp_buf_alloc(size_t len)</code>	36
4.15.2.5	<code>al_net_udp_buf_free(void *buf)</code>	36
4.15.2.6	<code>al_net_udp_connect(struct al_net_udp *udp, struct al_net_addr *addr, u16 port)</code>	36
4.15.2.7	<code>al_net_udp_free(struct al_net_udp *udp)</code>	36
4.15.2.8	<code>al_net_udp_new(void)</code>	37
4.15.2.9	<code>al_net_udp_send(struct al_net_udp *udp, void *buf, size_t len)</code>	37
4.15.2.10	<code>al_net_udp_sendto_if(struct al_net_udp *udp, void *buf, size_t len, struct al_net_addr *to, unsigned short port, struct al_net_if *nif)</code>	37

4.15.2.11	<code>al_net_udp_set_rcv_cb(struct al_net_udp *udp, void(*rcv_cb)(void *arg, struct al_net_udp *udp, void *data, size_t len, struct al_net_addr *from_ip, unsigned short from_port, struct al_net_if *net_if), void *rcv_arg)</code>	38
4.16	<code>al/al_os_lock.h</code> File Reference	38
4.16.1	Detailed Description	38
4.16.2	Function Documentation	38
4.16.2.1	<code>al_os_lock_create(void)</code>	38
4.16.2.2	<code>al_os_lock_destroy(struct al_lock *lock)</code>	38
4.16.2.3	<code>al_os_lock_lock(struct al_lock *lock)</code>	39
4.16.2.4	<code>al_os_lock_unlock(struct al_lock *lock)</code>	39
4.17	<code>al/al_os_mem.h</code> File Reference	39
4.17.1	Detailed Description	39
4.17.2	Enumeration Type Documentation	40
4.17.2.1	<code>al_os_mem_type</code>	40
4.17.3	Function Documentation	40
4.17.3.1	<code>al_os_mem_alloc(size_t size)</code>	40
4.17.3.2	<code>al_os_mem_calloc(size_t size)</code>	40
4.17.3.3	<code>al_os_mem_free(void *mem)</code>	40
4.17.3.4	<code>al_os_mem_get_type(void)</code>	41
4.17.3.5	<code>al_os_mem_set_type(enum al_os_mem_type type)</code>	41
4.18	<code>al/al_os_reboot.h</code> File Reference	41
4.18.1	Detailed Description	41
4.18.2	Function Documentation	41
4.18.2.1	<code>al_os_reboot(void)</code>	41
4.19	<code>al/al_os_thread.h</code> File Reference	41
4.19.1	Detailed Description	42
4.19.2	Enumeration Type Documentation	42
4.19.2.1	<code>al_os_thread_pri</code>	42
4.19.3	Function Documentation	42
4.19.3.1	<code>al_os_thread_create(const char *name, void *stack, size_t stack_size, enum al_os_thread_pri pri, void(*thread_main)(struct al_thread *thread, void *arg), void *arg)</code>	42

4.19.3.2	<code>al_os_thread_get_exit_flag(struct al_thread *thread)</code>	43
4.19.3.3	<code>al_os_thread_get_priority(struct al_thread *thread)</code>	43
4.19.3.4	<code>al_os_thread_join(struct al_thread *thread)</code>	43
4.19.3.5	<code>al_os_thread_resume(struct al_thread *thread)</code>	43
4.19.3.6	<code>al_os_thread_self(void)</code>	44
4.19.3.7	<code>al_os_thread_set_exit_code(struct al_thread *thread, u32 code)</code>	44
4.19.3.8	<code>al_os_thread_set_priority(struct al_thread *thread, enum al_os_thread_pri pri)</code>	44
4.19.3.9	<code>al_os_thread_sleep(int ms)</code>	44
4.19.3.10	<code>al_os_thread_suspend(struct al_thread *thread)</code>	45
4.19.3.11	<code>al_os_thread_terminate(struct al_thread *thread)</code>	45
4.19.3.12	<code>al_os_thread_terminate_with_status(struct al_thread *thread)</code>	45
4.20	<code>al/al_persist.h</code> File Reference	45
4.20.1	Detailed Description	46
4.20.2	Enumeration Type Documentation	46
4.20.2.1	<code>al_persist_section</code>	46
4.20.3	Function Documentation	46
4.20.3.1	<code>al_persist_data_erase(enum al_persist_section section)</code>	46
4.20.3.2	<code>al_persist_data_read(enum al_persist_section section, const char *name, void *buf, size_t len)</code>	47
4.20.3.3	<code>al_persist_data_write(enum al_persist_section section, const char *name, const void *buf, size_t len)</code>	47
4.21	<code>al/al_random.h</code> File Reference	47
4.21.1	Detailed Description	47
4.21.2	Function Documentation	48
4.21.2.1	<code>al_random_fill(void *buf, size_t len)</code>	48
4.22	<code>al/al_rsa.h</code> File Reference	48
4.22.1	Detailed Description	48
4.22.2	Function Documentation	48
4.22.2.1	<code>al_rsa_ctxt_alloc(void)</code>	48
4.22.2.2	<code>al_rsa_ctxt_free(struct al_rsa_ctxt *ctxt)</code>	48
4.22.2.3	<code>al_rsa_encrypt_pub(struct al_rsa_ctxt *ctxt, const void *in, size_t in_len, void *out, size_t out_len)</code>	49

4.22.2.4	<code>al_rsa_key_clear(struct al_rsa_ctxt *ctxt)</code>	49
4.22.2.5	<code>al_rsa_pub_key_set(struct al_rsa_ctxt *ctxt, const void *key, size_t keylen)</code> . . .	49
4.22.2.6	<code>al_rsa_verify(struct al_rsa_ctxt *ctxt, const void *in, size_t in_len, void *out, size_t out_len)</code>	50
4.23	<code>al/al_utypes.h</code> File Reference	50
4.23.1	Detailed Description	50
4.23.2	Typedef Documentation	50
4.23.2.1	<code>s16</code>	50
4.23.2.2	<code>s32</code>	51
4.23.2.3	<code>s64</code>	51
4.23.2.4	<code>s8</code>	51
4.23.2.5	<code>size_t</code>	51
4.23.2.6	<code>ssize_t</code>	51
4.23.2.7	<code>u16</code>	51
4.23.2.8	<code>u32</code>	51
4.23.2.9	<code>u64</code>	51
4.23.2.10	<code>u8</code>	51
	Index	53

Chapter 1

Class Index

1.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

al_ada_callback	5
al_net_dns_req	6

Chapter 2

File Index

2.1 File List

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

al/al_ada_thread.h	7
al/al_aes.h	9
al/al_assert.h	11
al/al_cli.h	12
al/al_clock.h	13
al/al_err.h	15
al/al_hash_sha1.h	17
al/al_hash_sha256.h	18
al/al_httpd.h	20
al/al_log.h	24
al/al_net_addr.h	25
al/al_net_dns.h	26
al/al_net_if.h	27
al/al_net_mdns.h	??
al/al_net_stream.h	30
al/al_net_udp.h	34
al/al_os_lock.h	38
al/al_os_mem.h	39
al/al_os_reboot.h	41
al/al_os_thread.h	41
al/al_persist.h	45
al/al_random.h	47
al/al_rsa.h	48
al/al_utypes.h	50

Chapter 3

Class Documentation

3.1 `al_ada_callback` Struct Reference

```
#include <al_ada_thread.h>
```

Public Attributes

- `u8 pending`
- `void(* func)(void *)`
- `void * arg`
- `struct al_ada_callback * next`

3.1.1 Detailed Description

The callback structure.

3.1.2 Member Data Documentation

3.1.2.1 `void* al_ada_callback::arg`

The argument is passed to callback function

3.1.2.2 `void(* al_ada_callback::func)(void *)`

The callback function

3.1.2.3 `struct al_ada_callback* al_ada_callback::next`

Link to next callback

3.1.2.4 u8 al_ada_callback::pending

The pending flag

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

- [al/al_ada_thread.h](#)

3.2 al_net_dns_req Struct Reference

```
#include <al_net_dns.h>
```

Public Attributes

- const char * [hostname](#)
- enum [al_err](#) [error](#)
- void(* [callback](#))(struct [al_net_dns_req](#) *)
- struct [al_net_addr](#) [addr](#)
- void * [al_priv](#)

3.2.1 Detailed Description

Structure representing a DNS request.

Note that this does not handle IPv6 at this point. The requester should allocate this so that it persists for the duration of the DNS request. Zero it and then fill in the hostname and callback function.

3.2.2 Member Data Documentation

3.2.2.1 struct [al_net_addr](#) [al_net_dns_req::addr](#)

result of network address

3.2.2.2 void* [al_net_dns_req::al_priv](#)

private pointer for adaptation layer

3.2.2.3 void(* [al_net_dns_req::callback](#))(struct [al_net_dns_req](#) *)

function for delivering result

3.2.2.4 enum [al_err](#) [al_net_dns_req::error](#)

result error code (or in-progress)

3.2.2.5 const char* [al_net_dns_req::hostname](#)

hostname being looked up

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

- [al/al_net_dns.h](#)

Chapter 4

File Documentation

4.1 al/al_ada_thread.h File Reference

```
#include <ayla/utypes.h>
```

Classes

- struct [al_ada_callback](#)

Functions

- void [al_ada_main_loop](#) (void)
- void [al_ada_wakeup](#) (void)
- void [al_ada_kill](#) (void)
- void [al_ada_callback_init](#) (struct [al_ada_callback](#) *cb, void(*func)(void *), void *arg)
- void [al_ada_call](#) (struct [al_ada_callback](#) *cb)
- void [al_ada_sync_call](#) (struct [al_ada_callback](#) *cb)
- void [al_ada_timer_set](#) (struct timer *timer, unsigned long ms)
- void [al_ada_timer_cancel](#) (struct timer *timer)

4.1.1 Detailed Description

ADA Thread Interfaces

The ADA thread may be combined with another thread, such as the LwIP TCP/IP thread, if desired.

4.1.2 Function Documentation

4.1.2.1 void [al_ada_call](#) (struct [al_ada_callback](#) * *cb*)

Call callback function.

The function can be called on any thread, it puts the callback structure into ada thread's callback queue, and then wakes up the ada thread, in ada thread main loop, the callback function is called.

Parameters

<i>cb</i>	is a callback structure.
-----------	--------------------------

4.1.2.2 void `al_ada_callback_init` (struct `al_ada_callback` * *cb*, void(*) (void *) *func*, void * *arg*)

Initialize the callback structure.

Parameters

<i>cb</i>	is a callback structure.
<i>func</i>	is a callback function.
<i>arg</i>	is a argument to pass to the callback function.

4.1.2.3 void `al_ada_kill` (void)

Stop the main ADA thread.

Indicate that the ADA thread should exit for cleanup. This would probably be used only in test programs. This does not wait for the ADA thread to exit before returning.

4.1.2.4 void `al_ada_main_loop` (void)

Enter the ADA thread main loop.

Call [al_ada_kill\(\)](#) to quit the main loop.

4.1.2.5 void `al_ada_sync_call` (struct `al_ada_callback` * *cb*)

Synchronized call callback function.

If the function is called on ada thread, it is a nested call callback function. If the function is called on other thread, it similars `al_ada_call`, but the callback function with higher priority and the caller thread is blocked until the callback returns.

Parameters

<i>cb</i>	is a callback structure.
-----------	--------------------------

4.1.2.6 void `al_ada_timer_cancel` (struct `timer` * *timer*)

Cancel the started timer.

Parameters

<i>timer</i>	points structure that is already initialized by timer_init.
--------------	---

4.1.2.7 void al_ada_timer_set (struct timer * *timer*, unsigned long *ms*)

Start the timer.

Parameters

<i>timer</i>	points a structure that is already initialized by timer_init.
<i>ms</i>	is delay in millisecond to trigger the callback.

4.1.2.8 void al_ada_wakeup (void)

Wakeup the main ADA thread.

4.2 al/al_aes.h File Reference

```
#include <al/al_utypes.h>
```

Functions

- struct al_aes_ctxt * [al_aes_ctxt_alloc](#) (void)
- void [al_aes_ctxt_free](#) (struct al_aes_ctxt *ctxt)
- int [al_aes_cbc_key_set](#) (struct al_aes_ctxt *ctxt, void *key, [size_t](#) key_len, void *iv, int decrypt)
- int [al_aes_iv_get](#) (struct al_aes_ctxt *ctxt, void *buf, [size_t](#) len)
- int [al_aes_cbc_encrypt](#) (struct al_aes_ctxt *ctxt, const void *in, void *out, [size_t](#) len)
- int [al_aes_cbc_decrypt](#) (struct al_aes_ctxt *ctxt, const void *in, void *out, [size_t](#) len)

4.2.1 Detailed Description

AES Cryptography Interfaces.

4.2.2 Function Documentation

4.2.2.1 int al_aes_cbc_decrypt (struct al_aes_ctxt * *ctxt*, const void * *in*, void * *out*, [size_t](#) *len*)

Decrypt a block of data with AES.

Parameters

<i>ctxt</i>	points to the AES context structure.
<i>in</i>	points to the input cipher-text data.
<i>out</i>	points to the output buffer to receive the clear-text data. It may be the same as in.
<i>len</i>	gives the length of the in and out buffers.

Returns

zero on success, non-zero on error.

4.2.2.2 `int al_aes_cbc_encrypt (struct al_aes_ctxt * ctxt, const void * in, void * out, size_t len)`

Encrypt a block of data with AES.

Parameters

<i>ctxt</i>	points to the AES context structure.
<i>in</i>	points to the input clear-text data.
<i>out</i>	points to the output buffer to receive the cipher-text. It may be the same as in.
<i>len</i>	gives the length of the in and out buffers.

Returns

zero on success, non-zero on error.

4.2.2.3 `int al_aes_cbc_key_set (struct al_aes_ctxt * ctxt, void * key, size_t key_len, void * iv, int decrypt)`

Initialize the AES context with the key and IV for CBC mode.

Parameters

<i>ctxt</i>	points to the AES context structure.
<i>key</i>	points to the binary AES key.
<i>key_len</i>	gives the key length in bytes.
<i>iv</i>	points to the initialization vector.
<i>decrypt</i>	should be non-zero if setting up for decryption.

Returns

zero on success, non-zero on error.

4.2.2.4 `struct al_aes_ctxt* al_aes_ctxt_alloc (void)`

Allocate an AES context.

The AES context is used in all AES operations and can be used multiple times until it is freed.

Returns

an AES context pointer or NULL on allocation failure.

4.2.2.5 void al_aes_ctxt_free (struct al_aes_ctxt * ctxt)

Free an AES context.

Parameters

<i>ctxt</i>	points to the AES context structure. It may be NULL.
-------------	--

4.2.2.6 int al_aes_iv_get (struct al_aes_ctxt * ctxt, void * buf, size_t len)

Extract the IV from the AES context.

Note: this should be possible on all platforms. For CBC, the IV is the last 16 bytes of ciphertext. The implementation may need to do extra work to keep that available.

Parameters

<i>ctxt</i>	points to the AES context structure.
<i>buf</i>	points to the buffer to receive the IV.
<i>len</i>	indicates the length of the buffer.

Returns

zero on success, non-zero on error.

4.3 al/al_assert.h File Reference

Functions

- void [al_assert_handle](#) (const char *file, int line)

4.3.1 Detailed Description

Assert Interfaces.

4.3.2 Function Documentation

4.3.2.1 void al_assert_handle (const char * file, int line)

Assert handle

It's called when the assert occurs. Normally it should show an alert message and then reboot after a short delay.

Parameters

<i>file</i>	is the file name where the assert occurs.
<i>line</i>	is the line number where the assert occurs.

4.4 a/al_cli.h File Reference

Functions

- void [al_cli_register](#) (const char *cmd, const char *help, void(*handler)(int argc, char **argv))
- void [al_cli_set_prompt](#) (const char *pmpt)
- void [al_cli_print](#) (const char *str)

4.4.1 Detailed Description

Platform cli registration Interfaces

4.4.2 Function Documentation

4.4.2.1 void al_cli_print (const char * str)

This Prints string on console for CLI commands.

If no \n is present, the line should get no \n or \r added by the platform. The platform should ensure the str is shown out even if the \n is not provided.

Parameters

<i>str</i>	is a line to output.
------------	----------------------

4.4.2.2 void al_cli_register (const char * cmd, const char * help, void(*) (int argc, char **argv) handler)

It registers CLI command handler into platform CLI system

Parameters

<i>cmd</i>	is command string.
<i>help</i>	is the hit information.
<i>handler</i>	is the command handler.

Note: the cli registration should be before enter the main loop, otherwise an assert is raised

4.4.2.3 void al_cli_set_prompt (const char * *pmpt*)

It changes the CLI command prompt

It is not necessary, most module can stub this function.

Parameters

<i>pmpt</i>	is propmpt.
-------------	-------------

4.5 al/al_clock.h File Reference

```
#include <al/al_utypes.h>
```

Enumerations

- enum [al_clock_src](#) {
[AL_CS_NONE](#) = 0, [AL_CS_MIN](#) = 0x1120, [AL_CS_DEF](#) = 0x1130, [AL_CS_PERSIST](#) = 0x1135,
[AL_CS_HTTP](#) = 0x113c, [AL_CS_LOCAL](#) = 0x1140, [AL_CS_MCU_LO](#) = 0x1250, [AL_CS_SERVER](#) =
0x1260,
[AL_CS_SNTP](#) = [AL_CS_SERVER](#), [AL_CS_NTP](#) = 0x1270, [AL_CS_MCU_HI](#) = 0x1280, [AL_CS_LIMIT](#) }

Functions

- void [al_clock_reset_src](#) (void)
- int [al_clock_set](#) (u32 timestamp, enum [al_clock_src](#) clock_src)
- u32 [al_clock_get](#) (enum [al_clock_src](#) *clock_src)
- u64 [al_clock_get_total_ms](#) (void)

4.5.1 Detailed Description

Platform clock Interfaces

4.5.2 Enumeration Type Documentation

4.5.2.1 enum al_clock_src

Time source codes. Larger numbers indicate more reliable clock sources. Do not change the existing numbers, for upgrade compatibility.

Enumerator

[AL_CS_NONE](#) never been set.

[AL_CS_MIN](#) The min source.

[AL_CS_DEF](#) Set to CLOCK_START.

AL_CS_PERSIST From persisted data store(flash/nvram)
AL_CS_HTTP From cloud via HTTP request.
AL_CS_LOCAL Set by internal web server.
AL_CS_MCU_LO Set by MCU. low priority.
AL_CS_SERVER Set using server time.
AL_CS_SNTP Set using SNTP, not kept in sync.
AL_CS_NTP Set using NTP.
AL_CS_MCU_HI Set by MCU. high priority.
AL_CS_LIMIT Must be last.

4.5.3 Function Documentation

4.5.3.1 u32 al_clock_get (enum al_clock_src * clock_src)

Get clock

Parameters

<i>clock_src</i>	is the buffer to retrieve the clock source, It can be NULL.
------------------	---

Returns

the time as the number of seconds since 1970-01-01 00:00 (UTC).

4.5.3.2 u64 al_clock_get_total_ms (void)

Get 64 bits system tick

Returns

the system tick since the system boots.

4.5.3.3 void al_clock_reset_src (void)

Reset clock source This function only can be used in test framework.

4.5.3.4 int al_clock_set (u32 timestamp, enum al_clock_src clock_src)

Set clock

Parameters

<i>timestamp</i>	is the time as the number of seconds since 1970-01-01 00:00 (UTC).
<i>clock_src</i>	is the time source

Returns

0 on success.

4.6 al_err.h File Reference

Macros

- `#define AL_ERR_STRINGS`

Enumerations

- enum `al_err` {
`AL_ERR_OK` = 0, `AL_ERR_BUF` = 1, `AL_ERR_ALLOC` = 2, `AL_ERR_ERR` = 3,
`AL_ERR_NOT_FOUND` = 4, `AL_ERR_INVALID_VAL` = 5, `AL_ERR_INVALID_TYPE` = 6, `AL_ERR_IN_PROGRESS` = 7,
`AL_ERR_BUSY` = 8, `AL_ERR_LEN` = 9, `AL_ERR_INVALID_STATE` = 10, `AL_ERR_TIMEOUT` = 11,
`AL_ERR_ABORT` = 12, `AL_ERR_RST` = 13, `AL_ERR_CLSD` = 14, `AL_ERR_NOTCONN` = 15,
`AL_ERR_INVALID_NAME` = 16, `AL_ERR_RDONLY` = 17, `AL_ERR_CERT_EXP` = 18, `AL_ERR_COUNT` }

Functions

- `const char * al_err_string` (enum `al_err` err)

4.6.1 Detailed Description

Error numbers.

4.6.2 Macro Definition Documentation

4.6.2.1 #define AL_ERR_STRINGS

Value:

```
{
    \
    [AL_ERR_OK] = "none",          \
    [AL_ERR_BUF] = "buf",          \
    [AL_ERR_ALLOC] = "alloc failed", \
    [AL_ERR_ERR] = "error",        \
    [AL_ERR_NOT_FOUND] = "not found", \
    [AL_ERR_INVALID_VAL] = "inv val", \
    [AL_ERR_INVALID_TYPE] = "inv type", \
    [AL_ERR_IN_PROGRESS] = "in progress", \
    [AL_ERR_BUSY] = "busy",        \
    [AL_ERR_LEN] = "len",          \
    [AL_ERR_INVALID_STATE] = "inv state", \
    [AL_ERR_TIMEOUT] = "timeout",  \
    [AL_ERR_ABORT] = "conn abrt",  \
    [AL_ERR_RST] = "conn reset",   \
    [AL_ERR_CLSD] = "conn closed", \
    [AL_ERR_NOTCONN] = "not conn", \
    [AL_ERR_INVALID_NAME] = "inv name", \
    [AL_ERR_RDONLY] = "read-only property", \
    [AL_ERR_CERT_EXP] = "cert time", \
}
```

Initializer for error strings array. Keep this in sync with the enum `al_err` definition.

4.6.3 Enumeration Type Documentation

4.6.3.1 enum al_err

Error numbers.

These numbers are used by the platform adaptation layer and ADA.

The numbers must not be changed because they correspond to the enum ada_err values, which are used by the application layer.

Enumerator

AL_ERR_OK no error
AL_ERR_BUF network buf shortage - retry later
AL_ERR_ALLOC resource shortage
AL_ERR_ERR non-specific error
AL_ERR_NOT_FOUND object (e.g., property) not found
AL_ERR_INVAL_VAL invalid value
AL_ERR_INVAL_TYPE invalid type
AL_ERR_IN_PROGRESS successfully started, but not finished
AL_ERR_BUSY another operation is in progress
AL_ERR_LEN invalid length
AL_ERR_INVAL_STATE called without correct prerequisites
AL_ERR_TIMEOUT operation timed out
AL_ERR_ABRT connection aborted
AL_ERR_RST connection reset
AL_ERR_CLSD connection closed
AL_ERR_NOTCONN not connected
AL_ERR_INVAL_NAME invalid property name
AL_ERR_RDONLY tried to set a read-only value
AL_ERR_CERT_EXP SSL certificate not valid due to time
AL_ERR_COUNT count of enums, unused as an error code

4.6.4 Function Documentation

4.6.4.1 const char* al_err_string (enum al_err err)

Lookup a human-readable definition of an error number.

Parameters

<i>err</i>	the error number or the negative of the error number.
------------	---

Returns

a pointer to a string describing the error.

4.7 al/al_hash_sha1.h File Reference

```
#include <al/al_utypes.h>
```

Macros

- #define [AL_HASH_SHA1_SIZE](#) 20

Functions

- struct al_hash_sha1_ctxt * [al_hash_sha1_ctxt_alloc](#) (void)
- void [al_hash_sha1_ctxt_free](#) (struct al_hash_sha1_ctxt *ctxt)
- void [al_hash_sha1_ctxt_init](#) (struct al_hash_sha1_ctxt *ctxt)
- void [al_hash_sha1_add](#) (struct al_hash_sha1_ctxt *ctxt, const void *buf, [size_t](#) len)
- void [al_hash_sha1_final](#) (struct al_hash_sha1_ctxt *ctxt, void *buf)

4.7.1 Detailed Description

SHA-1 Cryptographic Hash Interfaces.

4.7.2 Macro Definition Documentation

4.7.2.1 #define AL_HASH_SHA1_SIZE 20

Size of SHA-1 hash.

4.7.3 Function Documentation

4.7.3.1 void al_hash_sha1_add (struct al_hash_sha1_ctxt * ctxt, const void * buf, size_t len)

Accumulate a SHA-1 hash by incorporating the supplied buffer into the previously-computed hash.

Parameters

<i>ctxt</i>	points to the initialized SHA-1 context.
<i>buf</i>	points to the buffer to use as input for the hash.
<i>len</i>	is the length of the buffer.

4.7.3.2 struct al_hash_sha1_ctxt* al_hash_sha1_ctxt_alloc (void)

Allocate a SHA-1 context.

The SHA-1 context is used in all SHA-1 operations and can be used multiple times until it is freed.

Returns

a SHA-1 context pointer or NULL on allocation failure.

4.7.3.3 void al_hash_sha1_ctxt_free (struct al_hash_sha1_ctxt * *ctxt*)

Free a SHA-1 context.

Parameters

<i>ctxt</i>	points to the SHA-1 context structure. It may be NULL.
-------------	--

4.7.3.4 void al_hash_sha1_ctxt_init (struct al_hash_sha1_ctxt * *ctxt*)

Initialize a platform-defined structure to start accumulating a SHA-1 hash.

Parameters

<i>ctxt</i>	points to the context to be initialized. This can be allocated on the stack by the application.
-------------	---

4.7.3.5 void al_hash_sha1_final (struct al_hash_sha1_ctxt * *ctxt*, void * *buf*)

Retrieve the SHA-1 hash result

Puts the resulting 20-byte SHA-1 hash in the buffer.

Parameters

<i>ctxt</i>	points to the initialized SHA-1 context.
<i>buf</i>	points to the buffer to receive the 20-byte SHA-1 hash result.

4.8 al/al_hash_sha256.h File Reference

```
#include <al/al_utypes.h>
```

Macros

- #define [AL_HASH_SHA256_SIZE](#) 32

Functions

- struct al_hash_sha256_ctxt * [al_hash_sha256_ctxt_alloc](#) (void)
- void [al_hash_sha256_ctxt_free](#) (struct al_hash_sha256_ctxt **ctxt*)
- void [al_hash_sha256_ctxt_init](#) (struct al_hash_sha256_ctxt **ctxt*)
- void [al_hash_sha256_add](#) (struct al_hash_sha256_ctxt **ctxt*, const void **buf*, [size_t](#) *len*)
- void [al_hash_sha256_final](#) (struct al_hash_sha256_ctxt **ctxt*, void **buf*)

4.8.1 Detailed Description

SHA-256 Cryptographic Hash Interfaces.

4.8.2 Macro Definition Documentation

4.8.2.1 #define AL_HASH_SHA256_SIZE 32

Size of SHA-256 hash.

4.8.3 Function Documentation

4.8.3.1 void al_hash_sha256_add (struct al_hash_sha256_ctxt * *ctxt*, const void * *buf*, size_t *len*)

Accumulate a SHA-256 hash by incorporating the supplied buffer into the previously-computed hash.

Parameters

<i>ctxt</i>	points to the initialized SHA-256 context.
<i>buf</i>	points to the buffer to use as input for the hash.
<i>len</i>	is the length of the buffer.

4.8.3.2 struct al_hash_sha256_ctxt* al_hash_sha256_ctxt_alloc (void)

Allocate a SHA-256 context.

The SHA-1 context is used in all SHA-1 operations and can be used multiple times until it is freed.

Returns

an SHA-256 context pointer or NULL on allocation failure.

4.8.3.3 void al_hash_sha256_ctxt_free (struct al_hash_sha256_ctxt * *ctxt*)

Free a SHA-256 context.

Parameters

<i>ctxt</i>	points to the SHA-256 context structure. It may be NULL.
-------------	--

4.8.3.4 void al_hash_sha256_ctxt_init (struct al_hash_sha256_ctxt * *ctxt*)

Initialize a platform-defined structure to start accumulating a SHA-256 hash.

Parameters

<i>ctxt</i>	points to the context to be initialized. This can be allocated on the stack by the application.
-------------	---

4.8.3.5 void al_hash_sha256_final (struct al_hash_sha256_ctxt * *ctxt*, void * *buf*)

Retrieve the SHA-256 hash result

Puts the resulting 20-byte SHA-256 hash in the buffer.

Parameters

<i>ctxt</i>	points to the initialized SHA-256 context.
<i>buf</i>	points to the buffer to receive the 20-byte SHA-256 hash result.

4.9 al/al_httpd.h File Reference

```
#include <stdlib.h>
#include <ayla/utypes.h>
```

Enumerations

- enum [al_http_method](#) {
AL_REQ_BAD = 0, **AL_REQ_GET**, **AL_REQ_GET_HEAD**, **AL_REQ_POST**,
AL_REQ_PUT, **AL_REQ_DELETE** }

Functions

- void [al_httpd_init](#) (void)
- void [al_httpd_final](#) (void)
- int [al_httpd_start](#) (u16 port)
- void [al_httpd_stop](#) (void)
- int [al_httpd_reg_url_cb](#) (const char *url, enum [al_http_method](#) method, int(*cb)(struct al_httpd_conn *conn))
- const char * [al_httpd_get_method](#) (struct al_httpd_conn *conn)
- const char * [al_httpd_get_url](#) (struct al_httpd_conn *conn)
- const char * [al_httpd_get_url_arg](#) (struct al_httpd_conn *conn, const char *name)
- const char * [al_httpd_get_version](#) (struct al_httpd_conn *conn)
- const char * [al_httpd_get_req_header](#) (struct al_httpd_conn *conn, const char *name)
- int [al_httpd_read](#) (struct al_httpd_conn *conn, char *buf, [size_t](#) buf_size)
- int [al_httpd_response](#) (struct al_httpd_conn *conn, int status, const char *headers)
- int [al_httpd_write](#) (struct al_httpd_conn *conn, const char *data, [size_t](#) size)
- void [al_httpd_close_conn](#) (struct al_httpd_conn *conn)

4.9.1 Detailed Description

User interface of HTTPD (HTTP server).

HTTPD works in single thread mode. Actually it works in the ADA thread. After it is started, server app should call `al_httpd_reg_urls_handler()` to register handler. When a remote client connects to the HTTPD server, the connection is accepted and an `al_httpd_conn` structure is created. When a HTTP request is received, the request header is parsed by HTTPD, and the registered handler is called. The request payload is handled by the handler. The argument of the handler is pointer to A structure of `al_httpd_conn`.

4.9.2 Enumeration Type Documentation

4.9.2.1 enum `al_http_method`

HTTP method or request type.

4.9.3 Function Documentation

4.9.3.1 void `al_httpd_close_conn (struct al_httpd_conn * conn)`

Close httpd client connection.

Parameters

<i>conn</i>	pointer to connection context
-------------	-------------------------------

4.9.3.2 void `al_httpd_final (void)`

Finalize HTTPD module.

4.9.3.3 const char* `al_httpd_get_method (struct al_httpd_conn * conn)`

Get method ("GET", "POST") in the current request.

Parameters

<i>conn</i>	pointer to connection context.
-------------	--------------------------------

Returns

an pointer to the method string in the request header.

4.9.3.4 const char* `al_httpd_get_req_header (struct al_httpd_conn * conn, const char * name)`

Get an field in the request header.

Parameters

<i>conn</i>	pointer to connection context
<i>name</i>	name of the item. It is case sensitive.

Returns

a pointer to the filed content in the request header.

Note: if the name is "Date", and the received header line is "Date:xxxxxxx\r\n", then it returns a pointer to "xxxxxxx".

4.9.3.5 `const char* al_httpd_get_url (struct al_httpd_conn * conn)`

Get url in the current request.

Parameters

<i>conn</i>	pointer to connection context.
-------------	--------------------------------

Returns

an pointer to the url string in the request header. NULL is not found. Note: For an url like "http://my-query?a=123&b=hello", the returned url is "http://my-query". To get argument a and b, please call: val1 = al_httpd_get_arg(struct al_httpd_conn *conn, "a"); val2 = al_httpd_get_arg(struct al_httpd_conn *conn, "b");

4.9.3.6 `const char* al_httpd_get_url_arg (struct al_httpd_conn * conn, const char * name)`

Get argument value in the current request's url.

Parameters

<i>conn</i>	pointer to connection context.
<i>name</i>	argument name.

Returns

an pointer to the value string of the argument. NULL is not found.

4.9.3.7 `const char* al_httpd_get_version (struct al_httpd_conn * conn)`

Get HTTP protocol version in the current request.

Parameters

<i>conn</i>	pointer to connection context.
-------------	--------------------------------

Returns

an pointer to the http version string in request header.

4.9.3.8 void al_httpd_init (void)

Initialize HTTPD module.

4.9.3.9 int al_httpd_read (struct al_httpd_conn * conn, char * buf, size_t buf_size)

Read the payload in the current request.

Parameters

<i>conn</i>	pointer to connection context
<i>buf</i>	buffer for data read
<i>buf_size</i>	size of the buffer

Returns

the number of bytes read. 0 if reached the end of the data. Negative is error.

4.9.3.10 int al_httpd_reg_url_cb (const char * url, enum al_http_method method, int (*)(struct al_httpd_conn *conn) cb)

Register a callback for a specified url and method.

Parameters

<i>url</i>	HTTP url
<i>method</i>	Index of HTTP method.
<i>cb</i>	a callback function to handle the url request.

Returns

status code: 0 is success.

4.9.3.11 int al_httpd_response (struct al_httpd_conn * conn, int status, const char * headers)

Send HTTP response header to the client.

Parameters

<i>conn</i>	pointer to connection context
<i>status</i>	HTTP status code.
<i>headers</i>	string of response headers. The headers are separated by '\r\n'. The last header is ended in '\r\n' or '\0'. It can be NULL if no additional headers need to be specified.

Returns

status code: 0 is success.

4.9.3.12 `int al_httpd_start (u16 port)`

Start an HTTPD server. The HTTPD works in ADA thread.

Parameters

<i>port</i>	service port
-------------	--------------

Returns

status code: 0 is success

4.9.3.13 `void al_httpd_stop (void)`

Stop httpd server

4.9.3.14 `int al_httpd_write (struct al_httpd_conn * conn, const char * data, size_t size)`

Write HTTP response payload.

Parameters

<i>conn</i>	pointer to connection context
<i>data</i>	data to be written
<i>size</i>	size of the data

Returns

status code: 0 is success.

4.10 `al/al_log.h` File Reference**Functions**

- void [al_log_print](#) (const char *line)
- const char * [al_log_get_mod_name](#) (u8 mod_id)

4.10.1 Detailed Description

Platform Logging Interfaces

4.10.2 Function Documentation

4.10.2.1 `const char* al_log_get_mod_name (u8 mod_id)`

Get the log mod name.

Parameters

<i>mod_id</i>	is log module ID.
---------------	-------------------

4.10.2.2 `void al_log_print (const char * line)`

This prints a single line on the console.

Parameters

<i>line</i>	string to print.
-------------	------------------

The supplied single line will be NUL-terminated and will not have a newline character ('\n') at the end.

The output will be terminated with a carriage return and line feed, as appropriate.

Where possible, the adaptation layer or SDK should buffer output but if buffers are full, it should block the thread (and the entire system if necessary) until the output can be added to the buffer.

Discussion: This would be called only from the logging system in lib/ayla. The port may prefix the log line with something like "[ada]" or just emit it unchanged, or it may not emit logs at all under it's own control, on the final product. Ideally the port would never drop log lines.

4.11 al/al_net_addr.h File Reference

```
#include <al/al_utypes.h>
#include <platform/pfm_net_addr.h>
```

Functions

- [u32 al_net_addr_get_ipv4](#) (const struct al_net_addr *addr)
- void [al_net_addr_set_ipv4](#) (struct al_net_addr *addr, [u32](#) ip)

4.11.1 Detailed Description

Platform Network Address Interfaces

4.11.2 Function Documentation

4.11.2.1 `u32 al_net_addr_get_ipv4 (const struct al_net_addr * addr)`

Get IPv4 address from the network address structure.

Parameters

<i>addr</i>	is a pointer to network address structure.
-------------	--

Returns

the IPv4 address in host byte order.

4.11.2.2 void `al_net_addr_set_ipv4` (struct `al_net_addr` * *addr*, u32 *ip*)

Set IPv4 address into the network address structure

Parameters

<i>addr</i>	is a pointer to network address structure.
<i>ip</i>	is the IPv4 address in host byte order

4.12 al/al_net_dns.h File Reference

```
#include <al/al_utypes.h>
#include <al/al_err.h>
#include <al/al_net_addr.h>
```

Classes

- struct [al_net_dns_req](#)

Functions

- enum [al_err al_dns_req_ipv4_start](#) (struct [al_net_dns_req](#) *req)
- void [al_dns_req_cancel](#) (struct [al_net_dns_req](#) *req)
- void [al_net_dns_delete_host](#) (const char *hostname)
- void [al_net_dns_servers_rotate](#) (void)

4.12.1 Detailed Description

Platform Network DNS Interfaces

4.12.2 Function Documentation

4.12.2.1 void `al_dns_req_cancel` (struct `al_net_dns_req` * *req*)

Cancel an DNS request.

Parameters

<i>req</i>	the DNS request.
------------	------------------

4.12.2.2 enum al_err al_dns_req_ipv4_start (struct al_net_dns_req * *req*)

Request an IPv4 DNS lookup by hostname.

Parameters

<i>req</i>	the DNS request structure to use, possibly uninitialized.
------------	---

The callback is always made asynchronously in the main thread. The request may be cancelled. And the *req* should be valid before the `al_dns_req_cancel` is called. This call must not block.

The callback must be done in the same thread and may be synchronous (on error or cached IP result), but otherwise will be asynchronous. TBD: Since the callback must be in the same thread, this is OK, but if we used multiple threads, the callback would need to always asynch.

Returns

zero on success, possible error codes are TBD.

4.12.2.3 void al_net_dns_delete_host (const char * *hostname*)

Delete a host from the DNS lookup cache.

Parameters

<i>hostname</i>	the hostname.
-----------------	---------------

4.12.2.4 void al_net_dns_servers_rotate (void)

Switch to the next DNS server in sequence, if possible. Used in error recovery.

4.13 al/al_net_if.h File Reference

```
#include <al/al_utypes.h>
#include <al/al_net_addr.h>
```

Enumerations

- enum `al_net_if_type` { `AL_NET_IF_DEF`, `AL_NET_IF_STA`, `AL_NET_IF_AP`, `AL_NET_IF_MAX` }

Functions

- struct al_net_if * [al_get_net_if](#) (enum [al_net_if_type](#) type)
- u32 [al_net_if_get_ipv4](#) (struct al_net_if *net_if)
- u32 [al_net_if_get_netmask](#) (struct al_net_if *net_if)
- int [al_net_if_get_mac_addr](#) (struct al_net_if *net_if, u8 mac_addr[6])
- struct al_net_addr * [al_net_if_get_addr](#) (struct al_net_if *net_if)

4.13.1 Detailed Description

Platform Network Interface APIs

4.13.2 Enumeration Type Documentation

4.13.2.1 enum al_net_if_type

Network interface type.

Enumerator

- AL_NET_IF_DEF*** Default interface
- AL_NET_IF_STA*** Wifi station interface
- AL_NET_IF_AP*** Wifi AP interface
- AL_NET_IF_MAX*** Wifi AP interface limit

4.13.3 Function Documentation

4.13.3.1 struct al_net_if* al_get_net_if(enum al_net_if_type type)

Get network interface.

Parameters

<i>type</i>	specifies the network interface.
-------------	----------------------------------

Returns

the network interface, NULL is for no the interface.

4.13.3.2 struct al_net_addr* al_net_if_get_addr(struct al_net_if * net_if)

Get network address is associated with the network interface

Parameters

<i>net_if</i>	is a pointer to network interface structure.
---------------	--

Returns

the network address structure pointer.

4.13.3.3 u32 al_net_if_get_ipv4 (struct al_net_if * *net_if*)

Get IPv4 address is associated with the network interface

Parameters

<i>net_if</i>	is a pointer to network interface structure.
---------------	--

Returns

the IPv4 address in host byte order.

4.13.3.4 int al_net_if_get_mac_addr (struct al_net_if * *net_if*, u8 *mac_addr*[6])

Get MAC address is associated with the network interface

Parameters

<i>net_if</i>	is a pointer to network interface structure.
<i>mac_addr</i>	points a buffer to retrieve the MAC address.

Returns

zero on success, -1 on error.

4.13.3.5 u32 al_net_if_get_netmask (struct al_net_if * *net_if*)

Get netmask is associated with the network interface

Parameters

<i>net_if</i>	is a pointer to network interface structure.
---------------	--

Returns

the metmask in host byte order.

4.14 al/al_net_stream.h File Reference

```
#include <ayla/utypes.h>
#include <al/al_err.h>
#include <platform/pfm_net_addr.h>
```

Enumerations

- enum [al_net_stream_type](#) { [AL_NET_STREAM_TCP](#), [AL_NET_STREAM_TLS](#), [AL_NET_STREAM_MAX](#) }

Functions

- struct al_net_stream * [al_net_stream_new](#) (enum [al_net_stream_type](#) type)
- enum [al_err](#) [al_net_stream_close](#) (struct al_net_stream *stream)
- void [al_net_stream_set_arg](#) (struct al_net_stream *stream, void *arg)
- enum [al_err](#) [al_net_stream_connect](#) (struct al_net_stream *stream, const char *hostname, struct al_net_addr *host_addr, u16 port, enum [al_err](#) (*connected)(void *arg, struct al_net_stream *, enum [al_err](#) err))
- int [al_net_stream_is_established](#) (struct al_net_stream *stream)
- void [al_net_stream_continue_recv](#) (struct al_net_stream *stream)
- void [al_net_stream_set_recv_cb](#) (struct al_net_stream *stream, enum [al_err](#) (*recv_cb)(void *arg, struct al_net_stream *stream, void *data, [size_t](#) size))
- void [al_net_stream_recved](#) (struct al_net_stream *stream, [size_t](#) len)
- enum [al_err](#) [al_net_stream_write](#) (struct al_net_stream *stream, const void *data, [size_t](#) len)
- enum [al_err](#) [al_net_stream_output](#) (struct al_net_stream *stream)
- void [al_net_stream_set_sent_cb](#) (struct al_net_stream *stream, void (*sent_cb)(void *arg, struct al_net_stream *stream, [size_t](#) len_sent))
- void [al_net_stream_set_err_cb](#) (struct al_net_stream *stream, void (*err_cb)(void *arg, enum [al_err](#) err))
- struct pfm_net_tcp * [al_net_stream_get_tcp_obj](#) (struct al_net_stream *stream)

4.14.1 Detailed Description

Network stream interfaces

4.14.2 Enumeration Type Documentation**4.14.2.1 enum al_net_stream_type**

Type of stream connection.

Enumerator

[AL_NET_STREAM_TCP](#) Use unencrypted HTTP/1.1
[AL_NET_STREAM_TLS](#) Use HTTP/1.1 over TLS
[AL_NET_STREAM_MAX](#) al_net_stream_type limit

4.14.3 Function Documentation**4.14.3.1 enum al_err al_net_stream_close (struct al_net_stream * stream)**

Close the specified stream.

Parameters

<i>stream</i>	is the stream to be closed.
---------------	-----------------------------

4.14.3.2 `enum al_err al_net_stream_connect (struct al_net_stream * stream, const char * hostname, struct al_net_addr * host_addr, u16 port, enum al_err(*)(void *arg, struct al_net_stream *, enum al_err err) connected)`

Connect to remote server.

Parameters

<i>stream</i>	is the stream.
<i>hostname</i>	is the remote host name. It is used in certificates checking.
<i>host_addr</i>	is the remote host address.
<i>port</i>	is the remote port.
<i>connected</i>	is a callback which will called on connection ends. The first argument of <code>connected()</code> is the value set by <code>al_net_stream_set_arg()</code> . The second argument is a pointer to <code>al_net_stream</code> structure. The third argument is the error number.

Returns

zero on success, error code on failure.

4.14.3.3 `void al_net_stream_continue_rcv (struct al_net_stream * stream)`

continue to receive. If stream call the `rcv_cb`, it may pause receiving, and should call this function before `rcv_cb` returns.

Parameters

<i>stream</i>	is the stream.
---------------	----------------

4.14.3.4 `struct pfm_net_tcp* al_net_stream_get_tcp_obj (struct al_net_stream * stream)`

Get the associated net tcp object. The net tcp object can be used to call net tcp functions to set TCP options. Please do not use the object to send or receive data.

Parameters

<i>stream</i>	is the stream.
---------------	----------------

Returns

a pointer to `struct pfm_net_tcp`, or NULL on failure.

4.14.3.5 `int al_net_stream_is_established (struct al_net_stream * stream)`

Get the connection status of net stream.

Parameters

<i>stream</i>	is the stream.
---------------	----------------

Returns

non-zero if the connection is established (connected).

4.14.3.6 `struct al_net_stream* al_net_stream_new (enum al_net_stream_type type)`

Open a new stream.

Parameters

<i>type</i>	is the type of http client. It is also the stream type.
-------------	---

Returns

a pointer to struct al_net_stream, or NULL on failure.

4.14.3.7 `enum al_err al_net_stream_output (struct al_net_stream * stream)`

Send any queued data buffered in the stream.

Parameters

<i>stream</i>	is the stream.
---------------	----------------

Returns

zero on success, error code (TBD) on error.

This sends any data that has been buffered on the stream. This should be called after all the pfm_net_tcp_write() calls are complete for a request, for example, or when a response is complete. This call should not block waiting for output to be sent.

4.14.3.8 `void al_net_stream_recved (struct al_net_stream * stream, size_t len)`

Indicate that bytes have been received.

This call indicates that len bytes have been received and can be acknowledged by the stream. If the stack doesn't provide this functionality and automatically acknowledges any data received, this call can be a no-op.

Parameters

<i>stream</i>	is the stream.
<i>len</i>	is the number of bytes to be acknowledged.

4.14.3.9 void al_net_stream_set_arg (struct al_net_stream * *stream*, void * *arg*)

Set the callback argument for the stream.

Parameters

<i>stream</i>	is the stream.
<i>arg</i>	is the opaque argument to be passed to receive, accept, connected, and sent callbacks.

4.14.3.10 void al_net_stream_set_err_cb (struct al_net_stream * *stream*, void(*)(void *arg, enum al_err err) *err_cb*)

Set the error callback.

Parameters

<i>stream</i>	is the stream.
<i>err_cb</i>	is the callback for reporting error. The first argument of err_cb() is the value set by al_net_stream_set_arg() . The second argument is the error number.

4.14.3.11 void al_net_stream_set_rcv_cb (struct al_net_stream * *stream*, enum al_err(*)(void *arg, struct al_net_stream **stream*, void **data*, size_t size) *rcv_cb*)

Set the receive-callback for receiving data.

Parameters

<i>stream</i>	is the stream.
<i>rcv_cb</i>	is the callback for received data. The first argument of rcv_cb() is the value set by al_net_stream_set_arg() . The second argument is a pointer to the al_net_stream structure. The third argument is the data received, The fourth argument is the data size.

4.14.3.12 void al_net_stream_set_sent_cb (struct al_net_stream * *stream*, void(*)(void *arg, struct al_net_stream **stream*, size_t len_sent) *sent_cb*)

Set a callback to be called when data is sent to remote.

Parameters

<i>stream</i>	is the stream.
<i>sent_cb</i>	is a callback function to be called when data is sent and acknowledged.

The first argument of `sent_cb()` is the value set by `al_net_stream_set_arg()`. The second argument is stream handle. The third argument is the number of bytes sent.

4.14.3.13 `enum al_err al_net_stream_write (struct al_net_stream * stream, const void * data, size_t len)`

Put data to stream buffer. Actually transmission may or may not be delayed.

Parameters

<i>stream</i>	is the stream.
<i>data</i>	is the data to be sent.
<i>len</i>	is data size.

Returns

zero on success.

4.15 al/al_net_udp.h File Reference

```
#include <al/al_utypes.h>
#include <al/al_err.h>
#include <al/al_net_addr.h>
```

Functions

- `struct al_net_udp * al_net_udp_new (void)`
- `enum al_err al_net_udp_bind (struct al_net_udp *udp, struct al_net_addr *addr, u16 port)`
- `enum al_err al_net_udp_connect (struct al_net_udp *udp, struct al_net_addr *addr, u16 port)`
- `void al_net_udp_set_rcv_cb (struct al_net_udp *udp, void(*rcv_cb)(void *arg, struct al_net_udp *udp, void *data, size_t len, struct al_net_addr *from_ip, unsigned short from_port, struct al_net_if *net_if), void *rcv_arg)`
- `void * al_net_udp_buf_alloc (size_t len)`
- `void al_net_udp_buf_free (void *buf)`
- `enum al_err al_net_udp_send (struct al_net_udp *udp, void *buf, size_t len)`
- `enum al_err al_net_udp_sendto_if (struct al_net_udp *udp, void *buf, size_t len, struct al_net_addr *to, unsigned short port, struct al_net_if *nif)`
- `void al_net_udp_free (struct al_net_udp *udp)`
- `enum al_err al_net_igmp_joingroup (struct al_net_udp *udp, struct al_net_addr *if_addr, struct al_net_addr *group)`
- `enum al_err al_net_igmp_leavegroup (struct al_net_udp *udp, struct al_net_addr *if_addr, struct al_net_addr *group)`

4.15.1 Detailed Description

Platform Network UDP Interfaces

TBD, need text for overall considerations of threading and locking, etc.

4.15.2 Function Documentation

4.15.2.1 `enum al_err al_net_igmp_joingroup (struct al_net_udp * udp, struct al_net_addr * if_addr, struct al_net_addr * group)`

Join a UDP multicast group.

Parameters

<i>udp</i>	is the UDP PCB.
<i>if_addr</i>	is the network interface IP address.
<i>group</i>	is the IP multicast group address.

Returns

zero on success, error number on error.

4.15.2.2 `enum al_err al_net_igmp_leavegroup (struct al_net_udp * udp, struct al_net_addr * if_addr, struct al_net_addr * group)`

Leave a UDP multicast group.

Parameters

<i>udp</i>	is the UDP PCB.
<i>if_addr</i>	is the network interface IP address.
<i>group</i>	is the IP multicast group address.

Returns

zero on success, error number on error.

4.15.2.3 `enum al_err al_net_udp_bind (struct al_net_udp * udp, struct al_net_addr * addr, u16 port)`

Bind a UDP PCB to a local IP address and port.

Parameters

<i>udp</i>	is the UDP PCB.
<i>addr</i>	is the local address to use, or NULL if any address can be used.
<i>port</i>	is the UDP port number, in host order.

Returns

zero on success, error number on error.

4.15.2.4 void* al_net_udp_buf_alloc (size_t len)

Allocate a UDP packet buffer.

Parameters

<i>len</i>	is the size of the buffer in bytes.
------------	-------------------------------------

Returns

a pointer to data buffer on success, NULL on error.

Usage: For sending, upper layer uses [al_net_udp_buf_alloc\(\)](#) to allocate a packet buffer. The net-udp layer will use [al_net_udp_buf_free\(\)](#) to free it when it is sent out. For receiving, net-udp layer uses [al_net_udp_buf_alloc\(\)](#) to allocate a packet buffer. The upper layer should use [al_net_udp_buf_free\(\)](#) to free it.

4.15.2.5 void al_net_udp_buf_free (void * buf)

Free UDP packet buffer.

Parameters

<i>buf</i>	is a pointer to data buffer. If NULL, it is ignored.
------------	--

4.15.2.6 enum al_err al_net_udp_connect (struct al_net_udp * udp, struct al_net_addr * addr, u16 port)

Set the remote address and/or port for a UDP PCB PCB.

This sets the destination address for subsequent [al_net_udp_send\(\)](#) calls.

Parameters

<i>udp</i>	is the UDP PCB.
<i>addr</i>	is the remote address to use.
<i>port</i>	is the remote UDP port number, in host order.

Returns

zero on success, error number on error.

4.15.2.7 void al_net_udp_free (struct al_net_udp * udp)

Free a UDP PCB.

Frees the UDP PCB, cancelling any pending callbacks.

Parameters

<i>udp</i>	is the UDP PCB.
------------	-----------------

4.15.2.8 struct al_net_udp* al_net_udp_new (void)

Allocate a UDP PCB.

This allocates and returns a UDP PCB.

Returns

the UDP PCB or NULL if the allocation fails.

4.15.2.9 enum al_err al_net_udp_send (struct al_net_udp * *udp*, void * *buf*, size_t *len*)

Send a packet on the UDP PCB.

Parameters

<i>udp</i>	is the UDP PCB.
<i>buf</i>	is a pointer to the data to be sent. The caller should use al_net_udp_buf_alloc() to allocate the data buffer. It will be freed by net-udp when the packet has been sent out.
<i>len</i>	is length of the data to be sent.

Returns

zero on success, error number on error.

The UDP buffer is freed, even on error.

4.15.2.10 enum al_err al_net_udp_sendto_if (struct al_net_udp * *udp*, void * *buf*, size_t *len*, struct al_net_addr * *to*, unsigned short *port*, struct al_net_if * *nif*)

Sent a packet on the UDP PCB.

Parameters

<i>udp</i>	is the UDP PCB.
<i>buf</i>	is a pointer to the data to be sent. The caller should use al_net_udp_buf_alloc() to allocate the buffer. It will be freed by net-udp when it has been sent out.
<i>len</i>	is the length of the data to be sent.
<i>to</i>	is the IP address of the destination.
<i>port</i>	is the UDP destination port in host order.
<i>nif</i>	is the network interface to use. If <i>nif</i> is NULL, any interface with a route to the destination may be used.

Returns

zero on success, error number on error.

The UDP buffer is freed, even on error.

4.15.2.11 `void al_net_udp_set_rcv_cb (struct al_net_udp * udp, void(*) (void *arg, struct al_net_udp *udp, void *data, size_t len, struct al_net_addr *from_ip, unsigned short from_port, struct al_net_if *net_if) rcv_cb, void * rcv_arg)`

Set the receive callback for a UDP PCB.

Parameters

<i>udp</i>	is the UDP PCB.
<i>rcv_cb</i>	is a function that will be called when a packet is received on the UDP PCB.
<i>rcv_arg</i>	is the value to be passed as the first argument to the receive callback.

The arguments of *rcv_cb*: *arg*: It is the last parameter of `al_net_udp_set_rcv_cb()`. *udp*: It is the net-udp object which reports the received data. *data*: UDP packet received. It is allocated in net-udp. The upper layer (or data receiver) should use `al_net_udp_buf_free()` to free the memory. *len*: It is the length of the received data. *from_ip*: It is the source IP address that the packet comes from. *from_port*: It is the source port, in host order. *net_if*: It is the network interface pointer, or NULL if unknown.

4.16 al/al_os_lock.h File Reference

Functions

- struct al_lock * `al_os_lock_create` (void)
- void `al_os_lock_lock` (struct al_lock **lock*)
- void `al_os_lock_unlock` (struct al_lock **lock*)
- void `al_os_lock_destroy` (struct al_lock **lock*)

4.16.1 Detailed Description

Platform OS mutexes Interfaces

4.16.2 Function Documentation

4.16.2.1 `struct al_lock* al_os_lock_create (void)`

Create a lock

Returns

pointer to the locker structure or NULL on failure

4.16.2.2 `void al_os_lock_destroy (struct al_lock * lock)`

Destroy the lock

Parameters

<i>lock</i>	is a pointer to the lock.
-------------	---------------------------

4.16.2.3 void al_os_lock_lock (struct al_lock * *lock*)

Lock the lock

The function blocks the current thread when the lock is locked.

Parameters

<i>lock</i>	is a pointer to the lock.
-------------	---------------------------

4.16.2.4 void al_os_lock_unlock (struct al_lock * *lock*)

Unlock the lock

Parameters

<i>lock</i>	is a pointer to the lock.
-------------	---------------------------

4.17 al/al_os_mem.h File Reference

```
#include <al/al_utypes.h>
```

Enumerations

- enum [al_os_mem_type](#) { [al_os_mem_type_long_period](#), [al_os_mem_type_long_cache](#) }

Functions

- void [al_os_mem_set_type](#) (enum [al_os_mem_type](#) type)
- enum [al_os_mem_type](#) [al_os_mem_get_type](#) (void)
- void * [al_os_mem_alloc](#) (size_t size)
- void * [al_os_mem_calloc](#) (size_t size)
- void [al_os_mem_free](#) (void *mem)

4.17.1 Detailed Description

Platform OS memory management Interfaces

4.17.2 Enumeration Type Documentation

4.17.2.1 enum al_os_mem_type

Memory allocation type.

Enumerator

al_os_mem_type_long_period Long period memory

al_os_mem_type_long_cache Cache memory

4.17.3 Function Documentation

4.17.3.1 void* al_os_mem_alloc (size_t size)

Allocates memory.

The memory will be appropriately aligned for the strictest alignment required by the platform, but at least 4-byte aligned in any case.

Parameters

<i>size</i>	is memory size required, in bytes.
-------------	------------------------------------

Returns

NULL if not enough memory of the specified size is available

4.17.3.2 void* al_os_mem_calloc (size_t size)

Allocates memory.

Allocates memory just like al_os_mem_alloc but also zeros it.

Parameters

<i>size</i>	is memory size required, in bytes.
-------------	------------------------------------

Returns

NULL if not enough memory of the specified size is available

4.17.3.3 void al_os_mem_free (void * mem)

Frees memory.

Parameters

<i>mem</i>	points to the memory, may be NULL.
------------	------------------------------------

4.17.3.4 enum al_os_mem_type al_os_mem_get_type (void)

Get memory allocation type of the current thread

Returns

allocation type.

4.17.3.5 void al_os_mem_set_type (enum al_os_mem_type type)

Set memory allocation type of the current thread

The implementation can use the information to allocate memory from different pool for the following allocation on the current thread.

Parameters

<i>type</i>	is allocation type.
-------------	---------------------

4.18 al/al_os_reboot.h File Reference

Functions

- void [al_os_reboot](#) (void)

4.18.1 Detailed Description

OS reboot Interfaces.

4.18.2 Function Documentation

4.18.2.1 void al_os_reboot (void)

Reboot the system.

4.19 al/al_os_thread.h File Reference

```
#include <ayla/utypes.h>
```

Enumerations

- enum [al_os_thread_pri](#) { [al_os_thread_pri_high](#), [al_os_thread_pri_normal](#), [al_os_thread_pri_low](#) }

Functions

- struct al_thread * [al_os_thread_create](#) (const char *name, void *stack, [size_t](#) stack_size, enum [al_os_thread_pri](#) pri, void(*thread_main)(struct al_thread *thread, void *arg), void *arg)
- int [al_os_thread_suspend](#) (struct al_thread *thread)
- int [al_os_thread_resume](#) (struct al_thread *thread)
- enum [al_os_thread_pri](#) [al_os_thread_get_priority](#) (struct al_thread *thread)
- void [al_os_thread_set_priority](#) (struct al_thread *thread, enum [al_os_thread_pri](#) pri)
- void [al_os_thread_set_exit_code](#) (struct al_thread *thread, [u32](#) code)
- int [al_os_thread_get_exit_flag](#) (struct al_thread *thread)
- void [al_os_thread_terminate](#) (struct al_thread *thread)
- [u32](#) [al_os_thread_terminate_with_status](#) (struct al_thread *thread)
- [u32](#) [al_os_thread_join](#) (struct al_thread *thread)
- struct al_thread * [al_os_thread_self](#) (void)
- void [al_os_thread_sleep](#) (int ms)

4.19.1 Detailed Description

Platform OS thread Interfaces

4.19.2 Enumeration Type Documentation

4.19.2.1 enum al_os_thread_pri

Thread priority.

Enumerator

- [al_os_thread_pri_high](#)** High priority
- [al_os_thread_pri_normal](#)** Normal priority
- [al_os_thread_pri_low](#)** Low priority

4.19.3 Function Documentation

- 4.19.3.1 struct al_thread* [al_os_thread_create](#) (const char * *name*, void * *stack*, [size_t](#) *stack_size*, enum [al_os_thread_pri](#) *pri*, void(*) (struct al_thread *thread, void *arg) *thread_main*, void * *arg*)

Create a thread

Parameters

<i>name</i>	is the thread name.
<i>stack</i>	is the memory of the stack, it's NULL to allocate the memory from the heap.
<i>stack_size</i>	is the stack size in bytes.
<i>pri</i>	is the thread priority.
<i>thread_main</i>	is a function of the main entry of the thread, It's parameters are the thread, a pointer to the thread, and the arg is passed from the arg of al_os_thread_create .
<i>arg</i>	is the parameter is passed to the thread_main.

Returns

pointer to the thread structure or NULL on failure.

4.19.3.2 int al_os_thread_get_exit_flag (struct al_thread * *thread*)

Get exit flag of the specified thread Generally, it is used in the loop of the thread itself to determine to quit or not.

Parameters

<i>thread</i>	is a pointer to the thread.
---------------	-----------------------------

Returns

non zero means to exit the thread loop.

4.19.3.3 enum al_os_thread_pri al_os_thread_get_priority (struct al_thread * *thread*)

Get the priority of a thread

Parameters

<i>thread</i>	is a pointer to the thread.
---------------	-----------------------------

Returns

the priority

4.19.3.4 u32 al_os_thread_join (struct al_thread * *thread*)

Block the current thread to wait for a thread terminated

Parameters

<i>thread</i>	is a pointer to the thread.
---------------	-----------------------------

Returns

exit-code of the thread.

4.19.3.5 int al_os_thread_resume (struct al_thread * *thread*)

Resume a thread

Parameters

<i>thread</i>	is a pointer to the thread.
---------------	-----------------------------

Returns

0 if the thread is resumed, others on failure.

4.19.3.6 struct al_thread* al_os_thread_self (void)

Get the structure of the current thread.

Returns

pointer to the thread structure or NULL on failure.

4.19.3.7 void al_os_thread_set_exit_code (struct al_thread * thread, u32 code)

Set the exit code for the specified thread

Parameters

<i>thread</i>	is a pointer to the thread.
<i>code</i>	is the exit code.

4.19.3.8 void al_os_thread_set_priority (struct al_thread * thread, enum al_os_thread_pri pri)

Set the priority of a thread

Parameters

<i>thread</i>	is a pointer to the thread.
<i>pri</i>	is the priority.

4.19.3.9 void al_os_thread_sleep (int ms)

Suspends execution of the calling thread for (at least) specified milliseconds.

Parameters

<i>ms</i>	is suspends period in millisecond.
-----------	------------------------------------

4.19.3.10 int al_os_thread_suspend (struct al_thread * *thread*)

Suspend a thread

Parameters

<i>thread</i>	is a pointer to the thread.
---------------	-----------------------------

Returns

0 if the thread is suspended, others on failure.

4.19.3.11 void al_os_thread_terminate (struct al_thread * *thread*)

Terminate the specified thread Generally, it is called by another thread that want to terminated the specified thread. Current thread can use this function to terminated itself.

Parameters

<i>thread</i>	is a pointer to the thread to be terminated.
---------------	--

4.19.3.12 u32 al_os_thread_terminate_with_status (struct al_thread * *thread*)

Terminate the specified thread synchronously This function set exit-flag for the thread, wait the thread being terminated, get the thread's exit-code, free memory of the thread, at last return the exit-code. Do not use this function to terminate the current thread itself.

Parameters

<i>thread</i>	is a pointer to the thread to be terminated.
---------------	--

Returns

exit-code of the thread.

4.20 al/al_persist.h File Reference

```
#include <al/al_utypes.h>
#include <al/al_err.h>
```

Enumerations

- enum [al_persist_section](#) { [AL_PERSIST_STARTUP](#) = 0, [AL_PERSIST_FACTORY](#) = 1 }

Functions

- enum `al_err al_persist_data_write` (enum `al_persist_section` section, const char *name, const void *buf, `size_t` len)
- `size_t al_persist_data_read` (enum `al_persist_section` section, const char *name, void *buf, `size_t` len)
- enum `al_err al_persist_data_erase` (enum `al_persist_section` section)

4.20.1 Detailed Description

Persistent Data Interfaces.

These APIs manage data saved in persistent storage (e.g., flash or disk). Typically 16KB is more enough to store all the data required. The storage must be written in so to avoid losing data if a reset occurs at any time. This usually will require multiple copies. If flash-erasable memory is used, excessive erasing should be avoided by appending rather than erasing and re-writing data. Care must be taken not to rewrite data that is unchanged.

4.20.2 Enumeration Type Documentation

4.20.2.1 enum `al_persist_section`

Persistent data section.

This designates the type of configuration. Factory configuration is normally configuration set by the manufacturer of the module and the device and not normally erased after shipment.

Startup configuration is data saved by the application and the agent during Wi-Fi setup and normal operation, and is erased by a factory reset.

Enumerator

`AL_PERSIST_STARTUP` save for startup configuration

`AL_PERSIST_FACTORY` save for factory configuration

4.20.3 Function Documentation

4.20.3.1 enum `al_err al_persist_data_erase` (enum `al_persist_section` section)

Erase all persisted data for a section.

Parameters

<code>section</code>	the type of configuration being erased.
----------------------	---

Returns

zero on success and a negative error number on error.

This is done on a factory reset to forget all data persisted from the cloud and from Wi-Fi setup, etc.

4.20.3.2 `ssize_t al_persist_data_read (enum al_persist_section section, const char * name, void * buf, size_t len)`

Read persistent data.

Parameters

<i>section</i>	the type of configuration being read.
<i>name</i>	the name of the data item.
<i>buf</i>	a pointer to the buffer to receive the data.
<i>len</i>	the length of the buffer, in bytes.

Returns

the length of bytes read, negative means error.

4.20.3.3 `enum al_err al_persist_data_write (enum al_persist_section section, const char * name, const void * buf, size_t len)`

Write persistent data.

Parameters

<i>section</i>	the type of configuration being saved.
<i>name</i>	the name of the data item.
<i>buf</i>	a pointer to the arbitrary binary data.
<i>len</i>	the length of the data, in bytes. It may be zero to erase the item.

Returns

zero on success or a negative error number on error. Possible errors are AL_ERR_ERR on a hardware erase or write error.

4.21 al/al_random.h File Reference

```
#include <al/al_utypes.h>
```

Functions

- `int al_random_fill (void *buf, size_t len)`

4.21.1 Detailed Description

Random Number Generator Interfaces.

4.21.2 Function Documentation

4.21.2.1 `int al_random_fill (void * buf, size_t len)`

Fills the supplied buffer with random bytes.

This interface should initialize the RNG only if it has not already been initialized. The platform is expected to be able to supply sufficient entropy for the RNG. Consider using sources such as packet sizes and arrival intervals to increase entropy if needed.

Parameters

<i>buf</i>	the buffer to hold the random result.
<i>len</i>	the length of the buffer.

Returns

0 on success, and -1 if a random number generator is not available on the platform.

4.22 `al/al_rsa.h` File Reference

```
#include <al/al_utypes.h>
```

Functions

- `struct al_rsa_ctxt * al_rsa_ctxt_alloc (void)`
- `void al_rsa_ctxt_free (struct al_rsa_ctxt *ctxt)`
- `size_t al_rsa_pub_key_set (struct al_rsa_ctxt *ctxt, const void *key, size_t keylen)`
- `void al_rsa_key_clear (struct al_rsa_ctxt *ctxt)`
- `ssize_t al_rsa_encrypt_pub (struct al_rsa_ctxt *ctxt, const void *in, size_t in_len, void *out, size_t out_len)`
- `ssize_t al_rsa_verify (struct al_rsa_ctxt *ctxt, const void *in, size_t in_len, void *out, size_t out_len)`

4.22.1 Detailed Description

RSA Cryptography Interfaces.

4.22.2 Function Documentation

4.22.2.1 `struct al_rsa_ctxt* al_rsa_ctxt_alloc (void)`

Allocate an RSA context.

The RsA context is used in all RSA operations and can be used multiple times until it is freed.

Returns

an RSA context pointer or NULL on allocation failure.

4.22.2.2 `void al_rsa_ctxt_free (struct al_rsa_ctxt * ctxt)`

Free an RSA context.

Parameters

<i>ctxt</i>	points to the RSA context structure. It may be NULL.
-------------	--

4.22.2.3 `ssize_t al_rsa_encrypt_pub (struct al_rsa_ctxt * ctxt, const void * in, size_t in_len, void * out, size_t out_len)`

Encrypt using an RSA public key.

Parameters

<i>ctxt</i>	points to the RSA context, initialized with the public key.
<i>in</i>	points to the input buffer.
<i>in_len</i>	gives the input buffer length, in bytes. Note: <i>in_len</i> must be less than or equal to (RSA_key_size_in_bytes - 11).
<i>out</i>	points to the output buffer. If out is null, the return is the output buffer size required.
<i>out_len</i>	gives the output buffer length, in bytes.

Returns

the length of the buffer used, or -1 on error.

4.22.2.4 `void al_rsa_key_clear (struct al_rsa_ctxt * ctxt)`

Erase and free key material from the RSA context.

Parameters

<i>ctxt</i>	points to the RSA context to be uninitialized.
-------------	--

4.22.2.5 `size_t al_rsa_pub_key_set (struct al_rsa_ctxt * ctxt, const void * key, size_t keylen)`

Set RSA key from binary ASN-1 sequence buffer.

Parameters

<i>ctxt</i>	points to the RSA context to be initialized.
<i>key</i>	points to the public key in ASN-1 format.
<i>keylen</i>	is the length of the key in bytes.

Returns

RSA key length in bytes. 0 on failure.

The caller must call [al_rsa_key_clear\(\)](#), even on failure.

4.22.2.6 `ssize_t al_rsa_verify (struct al_rsa_ctxt * ctxt, const void * in, size_t in_len, void * out, size_t out_len)`

Perform an RSA verify operation, decrypting with the public key.

Parameters

<code>ctxt</code>	points to the RSA context, initialized with the public key.
<code>in</code>	points to the input buffer.
<code>in_len</code>	gives the input buffer length, in bytes. Note: <code>in_len</code> must be equal to the RSA key size in bytes.
<code>out</code>	points to the output buffer. If <code>out</code> is null, it's return value is the output buffer size required.
<code>out_len</code>	gives the output buffer length, in bytes. It must be greater than or equal to <code>(RSA_key_size_in_bytes - 11)</code> .

Returns

the length of the buffer used, or -1 on error.

4.23 `al/al_utypes.h` File Reference

```
#include <al/al_compiler.h>
```

Typedefs

- typedef unsigned char `u8`
- typedef unsigned short `u16`
- typedef unsigned long `u32`
- typedef signed char `s8`
- typedef short `s16`
- typedef long `s32`
- typedef unsigned long long `u64`
- typedef long long `s64`
- typedef unsigned long `size_t`
- typedef long `ssize_t`

4.23.1 Detailed Description

Platform types.

Portable code should use `<ayla/utypes.h>` which includes this file. The platform may override this with its own header.

TBD: we will change `u32/s32` to be `int` eventually.

4.23.2 Typedef Documentation

4.23.2.1 typedef short `s16`

signed 16-bit integer

4.23.2.2 typedef long s32

signed 32-bit integer

4.23.2.3 typedef long long s64

signed 64-bit integer

4.23.2.4 typedef signed char s8

signed 8-bit integer

4.23.2.5 typedef unsigned long size_t

unsigned size type

4.23.2.6 typedef long ssize_t

signed size type

4.23.2.7 typedef unsigned short u16

unsigned 16-bit integer

4.23.2.8 typedef unsigned long u32

unsigned 32-bit integer

4.23.2.9 typedef unsigned long long u64

unsigned 64-bit integer

4.23.2.10 typedef unsigned char u8

unsigned 8-bit integer

Index

AL_CS_DEF
 al_clock.h, [13](#)
AL_CS_HTTP
 al_clock.h, [14](#)
AL_CS_LIMIT
 al_clock.h, [14](#)
AL_CS_LOCAL
 al_clock.h, [14](#)
AL_CS_MCU_HI
 al_clock.h, [14](#)
AL_CS_MCU_LO
 al_clock.h, [14](#)
AL_CS_MIN
 al_clock.h, [13](#)
AL_CS_NONE
 al_clock.h, [13](#)
AL_CS_NTP
 al_clock.h, [14](#)
AL_CS_PERSIST
 al_clock.h, [13](#)
AL_CS_SERVER
 al_clock.h, [14](#)
AL_CS_SNTP
 al_clock.h, [14](#)
AL_ERR_ABRT
 al_err.h, [16](#)
AL_ERR_ALLOC
 al_err.h, [16](#)
AL_ERR_BUSY
 al_err.h, [16](#)
AL_ERR_BUF
 al_err.h, [16](#)
AL_ERR_CERT_EXP
 al_err.h, [16](#)
AL_ERR_CLSD
 al_err.h, [16](#)
AL_ERR_COUNT
 al_err.h, [16](#)
AL_ERR_ERR
 al_err.h, [16](#)
AL_ERR_IN_PROGRESS
 al_err.h, [16](#)
AL_ERR_INVALID_NAME
 al_err.h, [16](#)
AL_ERR_INVALID_STATE
 al_err.h, [16](#)
AL_ERR_INVALID_TYPE
 al_err.h, [16](#)
AL_ERR_INVALID_VAL
 al_err.h, [16](#)
AL_ERR_LEN
 al_err.h, [16](#)
AL_ERR_NOT_FOUND
 al_err.h, [16](#)
AL_ERR_NOTCONN
 al_err.h, [16](#)
AL_ERR_OK
 al_err.h, [16](#)
AL_ERR_RDONLY
 al_err.h, [16](#)
AL_ERR_RST
 al_err.h, [16](#)
AL_ERR_STRINGS
 al_err.h, [15](#)
AL_ERR_TIMEOUT
 al_err.h, [16](#)
AL_HASH_SHA1_SIZE
 al_hash_sha1.h, [17](#)
AL_HASH_SHA256_SIZE
 al_hash_sha256.h, [19](#)
AL_NET_IF_AP
 al_net_if.h, [28](#)
AL_NET_IF_DEF
 al_net_if.h, [28](#)
AL_NET_IF_MAX
 al_net_if.h, [28](#)
AL_NET_IF_STA
 al_net_if.h, [28](#)
AL_NET_STREAM_MAX
 al_net_stream.h, [30](#)
AL_NET_STREAM_TCP
 al_net_stream.h, [30](#)
AL_NET_STREAM_TLS
 al_net_stream.h, [30](#)
AL_PERSIST_FACTORY
 al_persist.h, [46](#)
AL_PERSIST_STARTUP
 al_persist.h, [46](#)
addr
 al_net_dns_req, [6](#)
al/al_ada_thread.h, [7](#)
al/al_aes.h, [9](#)
al/al_assert.h, [11](#)
al/al_cli.h, [12](#)
al/al_clock.h, [13](#)
al/al_err.h, [15](#)
al/al_hash_sha1.h, [17](#)
al/al_hash_sha256.h, [18](#)

- al/al_httpd.h, 20
- al/al_log.h, 24
- al/al_net_addr.h, 25
- al/al_net_dns.h, 26
- al/al_net_if.h, 27
- al/al_net_stream.h, 30
- al/al_net_udp.h, 34
- al/al_os_lock.h, 38
- al/al_os_mem.h, 39
- al/al_os_reboot.h, 41
- al/al_os_thread.h, 41
- al/al_persist.h, 45
- al/al_random.h, 47
- al/al_rsa.h, 48
- al/al_utypes.h, 50
- al_ada_call
 - al_ada_thread.h, 7
- al_ada_callback, 5
 - arg, 5
 - func, 5
 - next, 5
 - pending, 5
- al_ada_callback_init
 - al_ada_thread.h, 8
- al_ada_kill
 - al_ada_thread.h, 8
- al_ada_main_loop
 - al_ada_thread.h, 8
- al_ada_sync_call
 - al_ada_thread.h, 8
- al_ada_thread.h
 - al_ada_call, 7
 - al_ada_callback_init, 8
 - al_ada_kill, 8
 - al_ada_main_loop, 8
 - al_ada_sync_call, 8
 - al_ada_timer_cancel, 8
 - al_ada_timer_set, 9
 - al_ada_wakeup, 9
- al_ada_timer_cancel
 - al_ada_thread.h, 8
- al_ada_timer_set
 - al_ada_thread.h, 9
- al_ada_wakeup
 - al_ada_thread.h, 9
- al_aes.h
 - al_aes_cbc_decrypt, 9
 - al_aes_cbc_encrypt, 10
 - al_aes_cbc_key_set, 10
 - al_aes_ctxt_alloc, 10
 - al_aes_ctxt_free, 11
 - al_aes_iv_get, 11
- al_aes_cbc_decrypt
 - al_aes.h, 9
- al_aes_cbc_encrypt
 - al_aes.h, 10
- al_aes_cbc_key_set
 - al_aes.h, 10
- al_aes_ctxt_alloc
 - al_aes.h, 10
- al_aes_ctxt_free
 - al_aes.h, 11
- al_aes_iv_get
 - al_aes.h, 11
- al_assert.h
 - al_assert_handle, 11
- al_assert_handle
 - al_assert.h, 11
- al_cli.h
 - al_cli_print, 12
 - al_cli_register, 12
 - al_cli_set_prompt, 12
- al_cli_print
 - al_cli.h, 12
- al_cli_register
 - al_cli.h, 12
- al_cli_set_prompt
 - al_cli.h, 12
- al_clock.h
 - AL_CS_DEF, 13
 - AL_CS_HTTP, 14
 - AL_CS_LIMIT, 14
 - AL_CS_LOCAL, 14
 - AL_CS_MCU_HI, 14
 - AL_CS_MCU_LO, 14
 - AL_CS_MIN, 13
 - AL_CS_NONE, 13
 - AL_CS_NTP, 14
 - AL_CS_PERSIST, 13
 - AL_CS_SERVER, 14
 - AL_CS_SNTP, 14
 - al_clock_get, 14
 - al_clock_get_total_ms, 14
 - al_clock_reset_src, 14
 - al_clock_set, 14
 - al_clock_src, 13
- al_clock_get
 - al_clock.h, 14
- al_clock_get_total_ms
 - al_clock.h, 14
- al_clock_reset_src
 - al_clock.h, 14
- al_clock_set
 - al_clock.h, 14
- al_clock_src
 - al_clock.h, 13
- al_dns_req_cancel
 - al_net_dns.h, 26
- al_dns_req_ipv4_start
 - al_net_dns.h, 27
- al_err
 - al_err.h, 16
- al_err.h
 - AL_ERR_ABRT, 16
 - AL_ERR_ALLOC, 16
 - AL_ERR_BUSY, 16

- AL_ERR_BUF, 16
- AL_ERR_CERT_EXP, 16
- AL_ERR_CLSD, 16
- AL_ERR_COUNT, 16
- AL_ERR_ERR, 16
- AL_ERR_IN_PROGRESS, 16
- AL_ERR_INVALID_NAME, 16
- AL_ERR_INVALID_STATE, 16
- AL_ERR_INVALID_TYPE, 16
- AL_ERR_INVALID_VAL, 16
- AL_ERR_LEN, 16
- AL_ERR_NOT_FOUND, 16
- AL_ERR_NOTCONN, 16
- AL_ERR_OK, 16
- AL_ERR_RDONLY, 16
- AL_ERR_RST, 16
- AL_ERR_STRINGS, 15
- AL_ERR_TIMEOUT, 16
- al_err, 16
- al_err_string, 16
- al_err_string
 - al_err.h, 16
- al_get_net_if
 - al_net_if.h, 28
- al_hash_sha1.h
 - AL_HASH_SHA1_SIZE, 17
 - al_hash_sha1_add, 17
 - al_hash_sha1_ctxt_alloc, 17
 - al_hash_sha1_ctxt_free, 18
 - al_hash_sha1_ctxt_init, 18
 - al_hash_sha1_final, 18
- al_hash_sha1_add
 - al_hash_sha1.h, 17
- al_hash_sha1_ctxt_alloc
 - al_hash_sha1.h, 17
- al_hash_sha1_ctxt_free
 - al_hash_sha1.h, 18
- al_hash_sha1_ctxt_init
 - al_hash_sha1.h, 18
- al_hash_sha1_final
 - al_hash_sha1.h, 18
- al_hash_sha256.h
 - AL_HASH_SHA256_SIZE, 19
 - al_hash_sha256_add, 19
 - al_hash_sha256_ctxt_alloc, 19
 - al_hash_sha256_ctxt_free, 19
 - al_hash_sha256_ctxt_init, 19
 - al_hash_sha256_final, 20
- al_hash_sha256_add
 - al_hash_sha256.h, 19
- al_hash_sha256_ctxt_alloc
 - al_hash_sha256.h, 19
- al_hash_sha256_ctxt_free
 - al_hash_sha256.h, 19
- al_hash_sha256_ctxt_init
 - al_hash_sha256.h, 19
- al_hash_sha256_final
 - al_hash_sha256.h, 20
- al_http_method
 - al_httpd.h, 21
- al_httpd.h
 - al_http_method, 21
 - al_httpd_close_conn, 21
 - al_httpd_final, 21
 - al_httpd_get_method, 21
 - al_httpd_get_req_header, 21
 - al_httpd_get_url, 22
 - al_httpd_get_url_arg, 22
 - al_httpd_get_version, 22
 - al_httpd_init, 23
 - al_httpd_read, 23
 - al_httpd_reg_url_cb, 23
 - al_httpd_response, 23
 - al_httpd_start, 24
 - al_httpd_stop, 24
 - al_httpd_write, 24
- al_httpd_close_conn
 - al_httpd.h, 21
- al_httpd_final
 - al_httpd.h, 21
- al_httpd_get_method
 - al_httpd.h, 21
- al_httpd_get_req_header
 - al_httpd.h, 21
- al_httpd_get_url
 - al_httpd.h, 22
- al_httpd_get_url_arg
 - al_httpd.h, 22
- al_httpd_get_version
 - al_httpd.h, 22
- al_httpd_init
 - al_httpd.h, 23
- al_httpd_read
 - al_httpd.h, 23
- al_httpd_reg_url_cb
 - al_httpd.h, 23
- al_httpd_response
 - al_httpd.h, 23
- al_httpd_start
 - al_httpd.h, 24
- al_httpd_stop
 - al_httpd.h, 24
- al_httpd_write
 - al_httpd.h, 24
- al_log.h
 - al_log_get_mod_name, 25
 - al_log_print, 25
- al_log_get_mod_name
 - al_log.h, 25
- al_log_print
 - al_log.h, 25
- al_net_addr.h
 - al_net_addr_get_ipv4, 25
 - al_net_addr_set_ipv4, 26
- al_net_addr_get_ipv4
 - al_net_addr.h, 25

- al_net_addr_set_ipv4
 - al_net_addr.h, 26
- al_net_dns.h
 - al_dns_req_cancel, 26
 - al_dns_req_ipv4_start, 27
 - al_net_dns_delete_host, 27
 - al_net_dns_servers_rotate, 27
- al_net_dns_delete_host
 - al_net_dns.h, 27
- al_net_dns_req, 6
 - addr, 6
 - al_priv, 6
 - callback, 6
 - error, 6
 - hostname, 6
- al_net_dns_servers_rotate
 - al_net_dns.h, 27
- al_net_if.h
 - AL_NET_IF_AP, 28
 - AL_NET_IF_DEF, 28
 - AL_NET_IF_MAX, 28
 - AL_NET_IF_STA, 28
 - al_get_net_if, 28
 - al_net_if_get_addr, 28
 - al_net_if_get_ipv4, 29
 - al_net_if_get_mac_addr, 29
 - al_net_if_get_netmask, 29
 - al_net_if_type, 28
- al_net_if_get_addr
 - al_net_if.h, 28
- al_net_if_get_ipv4
 - al_net_if.h, 29
- al_net_if_get_mac_addr
 - al_net_if.h, 29
- al_net_if_get_netmask
 - al_net_if.h, 29
- al_net_if_type
 - al_net_if.h, 28
- al_net_igmp_joingroup
 - al_net_udp.h, 35
- al_net_igmp_leavegroup
 - al_net_udp.h, 35
- al_net_stream.h
 - AL_NET_STREAM_MAX, 30
 - AL_NET_STREAM_TCP, 30
 - AL_NET_STREAM_TLS, 30
 - al_net_stream_close, 30
 - al_net_stream_connect, 31
 - al_net_stream_continue_recv, 31
 - al_net_stream_get_tcp_obj, 31
 - al_net_stream_is_established, 31
 - al_net_stream_new, 32
 - al_net_stream_output, 32
 - al_net_stream_recved, 32
 - al_net_stream_set_arg, 33
 - al_net_stream_set_err_cb, 33
 - al_net_stream_set_recv_cb, 33
 - al_net_stream_set_sent_cb, 33
 - al_net_stream_type, 30
 - al_net_stream_write, 34
- al_net_stream_close
 - al_net_stream.h, 30
- al_net_stream_connect
 - al_net_stream.h, 31
- al_net_stream_continue_recv
 - al_net_stream.h, 31
- al_net_stream_get_tcp_obj
 - al_net_stream.h, 31
- al_net_stream_is_established
 - al_net_stream.h, 31
- al_net_stream_new
 - al_net_stream.h, 32
- al_net_stream_output
 - al_net_stream.h, 32
- al_net_stream_recved
 - al_net_stream.h, 32
- al_net_stream_set_arg
 - al_net_stream.h, 33
- al_net_stream_set_err_cb
 - al_net_stream.h, 33
- al_net_stream_set_recv_cb
 - al_net_stream.h, 33
- al_net_stream_set_sent_cb
 - al_net_stream.h, 33
- al_net_stream_type
 - al_net_stream.h, 30
- al_net_stream_write
 - al_net_stream.h, 34
- al_net_udp.h
 - al_net_igmp_joingroup, 35
 - al_net_igmp_leavegroup, 35
 - al_net_udp_bind, 35
 - al_net_udp_buf_alloc, 35
 - al_net_udp_buf_free, 36
 - al_net_udp_connect, 36
 - al_net_udp_free, 36
 - al_net_udp_new, 37
 - al_net_udp_send, 37
 - al_net_udp_sendto_if, 37
 - al_net_udp_set_recv_cb, 38
- al_net_udp_bind
 - al_net_udp.h, 35
- al_net_udp_buf_alloc
 - al_net_udp.h, 35
- al_net_udp_buf_free
 - al_net_udp.h, 36
- al_net_udp_connect
 - al_net_udp.h, 36
- al_net_udp_free
 - al_net_udp.h, 36
- al_net_udp_new
 - al_net_udp.h, 37
- al_net_udp_send
 - al_net_udp.h, 37
- al_net_udp_sendto_if
 - al_net_udp.h, 37

- al_net_udp_set_rcv_cb
 - al_net_udp.h, 38
- al_os_lock.h
 - al_os_lock_create, 38
 - al_os_lock_destroy, 38
 - al_os_lock_lock, 39
 - al_os_lock_unlock, 39
- al_os_lock_create
 - al_os_lock.h, 38
- al_os_lock_destroy
 - al_os_lock.h, 38
- al_os_lock_lock
 - al_os_lock.h, 39
- al_os_lock_unlock
 - al_os_lock.h, 39
- al_os_mem.h
 - al_os_mem_alloc, 40
 - al_os_mem_calloc, 40
 - al_os_mem_free, 40
 - al_os_mem_get_type, 41
 - al_os_mem_set_type, 41
 - al_os_mem_type, 40
 - al_os_mem_type_long_cache, 40
 - al_os_mem_type_long_period, 40
- al_os_mem_alloc
 - al_os_mem.h, 40
- al_os_mem_calloc
 - al_os_mem.h, 40
- al_os_mem_free
 - al_os_mem.h, 40
- al_os_mem_get_type
 - al_os_mem.h, 41
- al_os_mem_set_type
 - al_os_mem.h, 41
- al_os_mem_type
 - al_os_mem.h, 40
- al_os_mem_type_long_cache
 - al_os_mem.h, 40
- al_os_mem_type_long_period
 - al_os_mem.h, 40
- al_os_reboot
 - al_os_reboot.h, 41
- al_os_reboot.h
 - al_os_reboot, 41
- al_os_thread.h
 - al_os_thread_create, 42
 - al_os_thread_get_exit_flag, 43
 - al_os_thread_get_priority, 43
 - al_os_thread_join, 43
 - al_os_thread_pri, 42
 - al_os_thread_pri_high, 42
 - al_os_thread_pri_low, 42
 - al_os_thread_pri_normal, 42
 - al_os_thread_resume, 43
 - al_os_thread_self, 44
 - al_os_thread_set_exit_code, 44
 - al_os_thread_set_priority, 44
 - al_os_thread_sleep, 44
 - al_os_thread_suspend, 44
 - al_os_thread_terminate, 45
 - al_os_thread_terminate_with_status, 45
- al_persist.h
 - AL_PERSIST_FACTORY, 46
 - AL_PERSIST_STARTUP, 46
 - al_persist_data_erase, 46
 - al_persist_data_read, 46
 - al_persist_data_write, 47
 - al_persist_section, 46
- al_persist_data_erase
 - al_persist.h, 46
- al_persist_data_read
 - al_persist.h, 46
- al_persist_data_write
 - al_persist.h, 47
- al_persist_section
 - al_persist.h, 46
- al_priv
 - al_net_dns_req, 6
- al_random.h
 - al_random_fill, 48
- al_random_fill
 - al_random.h, 48
- al_rsa.h
 - al_rsa_ctxt_alloc, 48

- al_rsa_ctxt_free, [48](#)
 - al_rsa_encrypt_pub, [49](#)
 - al_rsa_key_clear, [49](#)
 - al_rsa_pub_key_set, [49](#)
 - al_rsa_verify, [49](#)
- al_rsa_ctxt_alloc
 - al_rsa.h, [48](#)
- al_rsa_ctxt_free
 - al_rsa.h, [48](#)
- al_rsa_encrypt_pub
 - al_rsa.h, [49](#)
- al_rsa_key_clear
 - al_rsa.h, [49](#)
- al_rsa_pub_key_set
 - al_rsa.h, [49](#)
- al_rsa_verify
 - al_rsa.h, [49](#)
- al_utypes.h
 - s16, [50](#)
 - s32, [50](#)
 - s64, [51](#)
 - s8, [51](#)
 - size_t, [51](#)
 - ssize_t, [51](#)
 - u16, [51](#)
 - u32, [51](#)
 - u64, [51](#)
 - u8, [51](#)
- arg
 - al_ada_callback, [5](#)
- callback
 - al_net_dns_req, [6](#)
- error
 - al_net_dns_req, [6](#)
- func
 - al_ada_callback, [5](#)
- hostname
 - al_net_dns_req, [6](#)
- next
 - al_ada_callback, [5](#)
- pending
 - al_ada_callback, [5](#)
- s16
 - al_utypes.h, [50](#)
- s32
 - al_utypes.h, [50](#)
- s64
 - al_utypes.h, [51](#)
- s8
 - al_utypes.h, [51](#)
- size_t
 - al_utypes.h, [51](#)
- ssize_t
 - al_utypes.h, [51](#)
- u16
 - al_utypes.h, [51](#)
- u32
 - al_utypes.h, [51](#)
- u64
 - al_utypes.h, [51](#)
- u8
 - al_utypes.h, [51](#)