lwIP 1.3.0

Version lwIP 1.3.0
3/23/2008 7:22 PM

Table of Contents

Introduction

lwIP is a small independent implementation of the TCP/IP protocol suite that has been developed by Adam Dunkels at the Computer and Networks Architectures (CNA) lab at the Swedish Institute of Computer Science (SICS).

The focus of the lwIP TCP/IP implementation is to reduce resource usage while still having a full scale TCP. This making lwIP suitable for use in embedded systems with tens of kilobytes of free RAM and room for around 40 kilobytes of code ROM.

lwIP features:

- `IP` (Internet Protocol) including packet forwarding over multiple network interfaces
- `ICMP` (Internet Control Message Protocol) for network maintenance and debugging
- `IGMP` (Internet Group Management Protocol) for multicast traffic management
- `UDP` (User Datagram Protocol) including experimental UDP-lite extensions
- `TCP` (Transmission Control Protocol) with congestion control, RTT estimation and fast recovery/fast retransmit
- `raw/native` API for enhanced performance
- `Optional` Berkeley-like socket API
- `DNS` (Domain names resolver)
- `SNMP` (Simple Network Management Protocol)
- `DHCP` (Dynamic Host Configuration Protocol)
- `AUTOIP` (for IPv4, conform with RFC 3927)
- `PPP` (Point-to-Point Protocol)
- `ARP` (Address Resolution Protocol) for Ethernet

Documentation

Development of lwIP is hosted on Savannah, a central point for software development, maintenance and distribution. Everyone can help improve lwIP by use of Savannah's interface, CVS and the mailing list. A core team of developers will commit changes to the CVS source tree.

http://savannah.nongnu.org/projects/lwip/

The original out-dated homepage of lwIP and Adam Dunkels' papers on lwIP are at the official lwIP home page:
http://www.sics.se/~adam/lwip/

Self documentation of the source code is regularly extracted from the current CVS sources and is available from this web page:
http://www.nongnu.org/lwip/

There is now a constantly growin wiki about lwIP at
http://lwip.scribblewiki.com/

Also, there are mailing lists you can subscribe at
http://savannah.nongnu.org/mail/?group=lwip
plus searchable archives:
http://lists.nongnu.org/archive/html/lwip-users/
http://lists.nongnu.org/archive/html/lwip-devel/

Reading Adam's papers, the files in docs/, browsing the source code documentation and browsing the mailing list archives is a good way to become familiar with the design of lwIP.

lwIP 1.3.0 Directory Hierarchy

lwIP 1.3.0 Directories

This directory hierarchy is sorted roughly, but not completely, alphabetically:

lwIP 1.3.0 Data Structure Index

lwIP 1.3.0 Data Structures

Here are the data structures with brief descriptions:

lwIP 1.3.0 File Index

lwIP 1.3.0 File List

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

lwIP 1.3.0 Page Index

lwIP 1.3.0 Related Pages

Here is a list of all related documentation pages:

lwIP 1.3.0 Directory Documentation

lwip/src/api/ Directory Reference

Files

- file **api_lib.c**
- file **api_msg.c**
- file **err.c**
- file **netbuf.c**
- file **netdb.c**
- file **netifapi.c**
- file **sockets.c**
- file **tcpip.c**

lwip/src/core/ Directory Reference

Directories

- directory **ipv4**
- directory **ipv6**
- directory **snmp**

Files

- file **dhcp.c**
- file **dns.c**
- file **init.c**
- file **mem.c**
- file **memp.c**
- file **netif.c**
- file **pbuf.c**
- file **raw.c**
- file **stats.c**
- file **sys.c**
- file **tcp.c**
- file **tcp_in.c**
- file **tcp_out.c**
- file **udp.c**

lwip/src/include/ Directory Reference

Directories

- directory **ipv4**
- directory **ipv6**
- directory **lwip**
- directory **netif**

lwip/src/include/ipv4/ Directory Reference

Directories

- directory **lwip**

lwip/src/core/ipv4/ Directory Reference

Files

- file **autoip.c**
- file **icmp.c**
- file **igmp.c**
- file **inet.c**
- file **inet_chksum.c**
- file **ip.c**
- file **ip_addr.c**
- file **ip_frag.c**

lwip/src/include/ipv6/ Directory Reference

Directories

- directory **lwip**

lwip/src/core/ipv6/ Directory Reference

Files

- file **icmp6.c**
- file **inet6.c**
- file **ip6.c**
- file **ip6_addr.c**

lwip/src/include/lwip/ Directory Reference

Files

- file **api.h**
- file **api_msg.h**
- file **arch.h**
- file **debug.h**
- file **def.h**
- file **dhcp.h**
- file **dns.h**
- file **err.h**
- file **init.h**
- file **mem.h**
- file **memp.h**

- file **memp_std.h**
- file **netbuf.h**
- file **netdb.h**
- file **netif.h**
- file **netifapi.h**
- file **opt.h**
- file **pbuf.h**
- file **raw.h**
- file **sio.h**
- file **snmp.h**
- file **snmp_asn1.h**
- file **snmp_msg.h**
- file **snmp_structs.h**
- file **sockets.h**
- file **stats.h**
- file **sys.h**
- file **tcp.h**
- file **tcpip.h**
- file **udp.h**

lwip/src/include/ipv6/lwip/ Directory Reference

Files

- file **icmp.h**
- file **inet.h**
- file **ip.h**
- file **ip_addr.h**

lwip/src/include/ipv4/lwip/ Directory Reference

Files

- file **autoip.h**
- file **icmp.h**
- file **igmp.h**
- file **inet.h**
- file **inet_chksum.h**
- file **ip.h**
- file **ip_addr.h**
- file **ip_frag.h**

lwip/ Directory Reference

Directories

- directory **src**

Files

- file **main_page.h**

lwip/src/netif/ Directory Reference

Directories

- directory **ppp**

Files

- file **etharp.c**
- file **ethernetif.c**
- file **loopif.c**
- file **slipif.c**

lwip/src/include/netif/ Directory Reference

Files

- file **etharp.h**
- file **loopif.h**
- file **ppp_oe.h**
- file **slipif.h**

lwip/src/netif/ppp/ Directory Reference

Files

- file **auth.c**
- file **auth.h**
- file **chap.c**
- file **chap.h**
- file **chpms.c**
- file **chpms.h**
- file **fsm.c**
- file **fsm.h**
- file **ipcp.c**
- file **ipcp.h**
- file **lcp.c**
- file **lcp.h**
- file **magic.c**
- file **magic.h**
- file **md5.c**
- file **md5.h**
- file **pap.c**
- file **pap.h**
- file **ppp.c**
- file **ppp.h**
- file **ppp_oe.c**

- file **pppdebug.h**
- file **randm.c**
- file **randm.h**
- file **vj.c**
- file **vj.h**
- file **vjbsdhdr.h**

lwip/src/core/snmp/ Directory Reference

Files

- file **asn1_dec.c**
- file **asn1_enc.c**
- file **mib2.c**
- file **mib_structs.c**
- file **msg_in.c**
- file **msg_out.c**

lwip/src/ Directory Reference

Directories

- directory **api**
- directory **core**
- directory **include**
- directory **netif**

lwIP 1.3.0 Data Structure Documentation

api_msg Struct Reference

`#include <lwip/src/include/lwip/api_msg.h>`

Data Fields

- void(* **function**)(struct **api_msg_msg** *__msg__)
- struct **api_msg_msg msg**

Detailed Description

This struct contains a function to execute in another thread context and a struct **api_msg_msg** that serves as an argument for this function. This is passed to tcpip_apimsg to execute functions in tcpip_thread context.

Field Documentation

void(* api_msg::function)(struct api_msg_msg *msg)

function to execute in tcpip_thread context

struct api_msg_msg api_msg::msg `[read]`

arguments for this function

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

- lwip/src/include/lwip/api_msg.h

api_msg_msg Struct Reference

`#include <lwip/src/include/lwip/api_msg.h>`

Data Fields

- struct **netconn** * **conn**
- struct netbuf * **b**

Detailed Description

This struct includes everything that is necessary to execute a function for a **netconn** in another thread context (mainly used to process netconns in the tcpip_thread context to be thread safe).

Field Documentation

struct netconn* api_msg_msg::conn `[read]`

The **netconn** which to process - always needed: it includes the semaphore which is used to block the application thread until the function finished.

struct netbuf* api_msg_msg::b `[read]`

used for do_send

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

- lwip/src/include/lwip/api_msg.h

dhcp_msg Struct Reference

`#include <lwip/src/include/lwip/dhcp.h>`

Detailed Description

minimum set of fields of any DHCP message

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

- lwip/src/include/lwip/**dhcp.h**

dns_answer Struct Reference

Detailed Description

DNS answer message structure

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

- lwip/src/core/**dns.c**

dns_api_msg Struct Reference

`#include <lwip/src/include/lwip/api_msg.h>`

Data Fields

- const char * **name**
- struct ip_addr * **addr**
- sys_sem_t **sem**
- err_t * **err**

Detailed Description

As do_gethostbyname requires more arguments but doesn't require a **netconn**, it has its own struct (to avoid struct **api_msg** getting bigger than necessary). do_gethostbyname must be called using tcpip_callback instead of tcpip_apimsg (see netconn_gethostbyname).

Field Documentation

const char* dns_api_msg::name

Hostname to query or dotted IP address string

struct ip_addr* dns_api_msg::addr `[read]`

Rhe resolved address is stored here

sys_sem_t dns_api_msg::sem

This semaphore is posted when the name is resolved, the application thread should wait on it.

err_t* dns_api_msg::err

Errors are given back here

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

- lwip/src/include/lwip/api_msg.h

dns_hdr Struct Reference

Detailed Description

DNS message header

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

- lwip/src/core/**dns.c**

dns_query Struct Reference

Detailed Description

DNS query message structure

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

- lwip/src/core/**dns.c**

dns_table_entry Struct Reference

Detailed Description

DNS table entry

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

- lwip/src/core/**dns.c**

etharp_hdr Struct Reference

`#include <lwip/src/include/netif/etharp.h>`

Detailed Description

the ARP message

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

- lwip/src/include/netif/etharp.h

etharp_q_entry Struct Reference

`#include <lwip/src/include/netif/etharp.h>`

Detailed Description

struct for queueing outgoing packets for unknown address defined here to be accessed by memp.h

---

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

- lwip/src/include/netif/etharp.h

ethernetif Struct Reference

Detailed Description

Helper struct to hold private data used to operate your ethernet interface. Keeping the ethernet address of the MAC in this struct is not necessary as it is already kept in the struct **netif**. But this is only an example, anyway...

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

- lwip/src/netif/**ethernetif.c**

gethostbyname_r_helper Struct Reference

Detailed Description

helper struct for gethostbyname_r to access the char* buffer

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

- lwip/src/api/**netdb.c**

ip_reass_helper Struct Reference

Detailed Description

This is a helper struct which holds the starting offset and the ending offset of this fragment to easily chain the fragments.

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

- lwip/src/core/ipv4/**ip_frag.c**

lwip_select_cb Struct Reference

Data Fields

- struct **lwip_select_cb** * **next**
- fd_set * **readset**
- fd_set * **writeset**
- fd_set * **exceptset**
- int **sem_signalled**
- sys_sem_t **sem**

Detailed Description

Description for a task waiting in select

Field Documentation

struct lwip_select_cb* lwip_select_cb::next `[read]`

Pointer to the next waiting task

fd_set* lwip_select_cb::readset

readset passed to select

fd_set* lwip_select_cb::writeset

writeset passed to select

fd_set* lwip_select_cb::exceptset

unimplemented: exceptset passed to select

int lwip_select_cb::sem_signalled

don't signal the same semaphore twice: set to 1 when signalled

sys_sem_t lwip_select_cb::sem

semaphore to wake up a task waiting for select

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

- lwip/src/api/**sockets.c**

lwip_setgetsockopt_data Struct Reference

Data Fields

- struct **lwip_socket** * **sock**
- int **s**
- int **level**
- int **optname**
- void * **optval**
- socklen_t * **optlen**
- err_t **err**

Detailed Description

This struct is used to pass data to the set/getsockopt_internal functions running in tcpip_thread context (only a void* is allowed)

Field Documentation

struct lwip_socket* lwip_setgetsockopt_data::sock `[read]`

socket struct for which to change options

int lwip_setgetsockopt_data::s

socket index for which to change options

int lwip_setgetsockopt_data::level

level of the option to process

int lwip_setgetsockopt_data::optname

name of the option to process

void* lwip_setgetsockopt_data::optval

set: value to set the option to get: value of the option is stored here

socklen_t* lwip_setgetsockopt_data::optlen

size of *optval

err_t lwip_setgetsockopt_data::err

if an error occures, it is temporarily stored here

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

- lwip/src/api/**sockets.c**

lwip_socket Struct Reference

Data Fields

- struct **netconn** * **conn**
- struct netbuf * **lastdata**
- u16_t **lastoffset**
- u16_t **rcvevent**
- u16_t **sendevent**
- u16_t **flags**
- int **err**

Detailed Description

Contains all internal pointers and states used for a socket

Field Documentation

struct netconn* lwip_socket::conn `[read]`

sockets currently are built on netconns, each socket has one **netconn**

struct netbuf* lwip_socket::lastdata `[read]`

data that was left from the previous read

u16_t lwip_socket::lastoffset

offset in the data that was left from the previous read

u16_t lwip_socket::rcvevent

number of times data was received, set by event_callback(), tested by the receive and select functions

u16_t lwip_socket::sendevent

number of times data was received, set by event_callback(), tested by select

u16_t lwip_socket::flags

socket flags (currently, only used for O_NONBLOCK)

int lwip_socket::err

last error that occurred on this socket

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

- lwip/src/api/**sockets.c**

mem Struct Reference

Data Fields

- mem_size_t **next**
- mem_size_t **prev**
- u8_t **used**

Detailed Description

The heap is made up as a list of structs of this type. This does not have to be aligned since for getting its size, we only use the macro SIZEOF_STRUCT_MEM, which automatically alignes.

Field Documentation

mem_size_t mem::next

index (-> ram[next]) of the next struct

mem_size_t mem::prev

index (-> ram[next]) of the next struct

u8_t mem::used

1: this area is used; 0: this area is unused

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

- lwip/src/core/**mem.c**

mem_helper Struct Reference

Detailed Description

This structure is used to save the pool one element came from.

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

- lwip/src/core/**mem.c**

mib_array_node Struct Reference

`#include <lwip/src/include/lwip/snmp_structs.h>`

Detailed Description

derived node, points to a fixed size const array of sub-identifiers plus a 'child' pointer

---

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

- lwip/src/include/lwip/**snmp_structs.h**

mib_external_node Struct Reference

`#include <lwip/src/include/lwip/snmp_structs.h>`

Data Fields

- void * **addr_inf**
- u8_t **tree_levels**
- u16_t(* **level_length**)(void ***addr_inf**, u8_t level)
- s32_t(* **ident_cmp**)(void ***addr_inf**, u8_t level, u16_t idx, s32_t sub_id)
- void(* **get_object_def_q**)(void ***addr_inf**, u8_t rid, u8_t ident_len, s32_t *ident)
- void(* **get_object_def_a**)(u8_t rid, u8_t ident_len, s32_t *ident, struct **obj_def** *od)
- void(* **get_object_def_pc**)(u8_t rid, u8_t ident_len, s32_t *ident)

Detailed Description

derived node, has access functions for mib object in external memory or device using 'tree_level' and 'idx', with a range 0 .. (**level_length()** - 1)

Field Documentation

void* mib_external_node::addr_inf

points to an extenal (in memory) record of some sort of addressing information, passed to and interpreted by the funtions below

u8_t mib_external_node::tree_levels

tree levels under this node

u16_t(* mib_external_node::level_length)(void *addr_inf, u8_t level)

number of objects at this level

s32_t(* mib_external_node::ident_cmp)(void *addr_inf, u8_t level, u16_t idx, s32_t sub_id)

compares object sub identifier with external id return zero when equal, nonzero when unequal

void(* mib_external_node::get_object_def_q)(void *addr_inf, u8_t rid, u8_t ident_len, s32_t *ident)

async Questions

void(* mib_external_node::get_object_def_a)(u8_t rid, u8_t ident_len, s32_t *ident, struct obj_def *od)

async Answers

void(* mib_external_node::get_object_def_pc)(u8_t rid, u8_t ident_len, s32_t *ident)

async Panic Close (agent returns error reply, e.g. used for external transaction cleanup)

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

- lwip/src/include/lwip/**snmp_structs.h**

mib_list_rootnode Struct Reference

`#include <lwip/src/include/lwip/snmp_structs.h>`

Detailed Description

derived node, points to a doubly linked list of sub-identifiers plus a 'child' pointer

---

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

- lwip/src/include/lwip/**snmp_structs.h**

mib_node Struct Reference

`#include <lwip/src/include/lwip/snmp_structs.h>`

Data Fields

- void(* **get_object_def**)(u8_t ident_len, s32_t *ident, struct **obj_def** *od)
- void(* **get_value**)(struct **obj_def** *od, u16_t len, void *value)
- u8_t(* **set_test**)(struct **obj_def** *od, u16_t len, void *value)
- void(* **set_value**)(struct **obj_def** *od, u16_t len, void *value)
- const u8_t **node_type**

Detailed Description

node "base class" layout, the mandatory fields for a node

Field Documentation

void(* mib_node::get_object_def)(u8_t ident_len, s32_t *ident, struct obj_def *od)

returns struct **obj_def** for the given object identifier

void(* mib_node::get_value)(struct obj_def *od, u16_t len, void *value)

returns object value for the given object identifier,

Note:

the caller must allocate at least len bytes for the value

u8_t(* mib_node::set_test)(struct obj_def *od, u16_t len, void *value)

tests length and/or range BEFORE setting

void(* mib_node::set_value)(struct obj_def *od, u16_t len, void *value)

sets object value, only to be called when **set_test()**

const u8_t mib_node::node_type

One out of MIB_NODE_AR, MIB_NODE_LR or MIB_NODE_EX

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

- lwip/src/include/lwip/**snmp_structs.h**

mib_ram_array_node Struct Reference

`#include <lwip/src/include/lwip/snmp_structs.h>`

Detailed Description

derived node, points to a fixed size mem_malloced array of sub-identifiers plus a 'child' pointer

---

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

- lwip/src/include/lwip/**snmp_structs.h**

netconn Struct Reference

`#include <lwip/src/include/lwip/api.h>`

Data Fields

- enum netconn_type **type**
- enum netconn_state **state**
- err_t **err**
- sys_sem_t **op_completed**
- sys_mbox_t **recvmbox**
- sys_mbox_t **acceptmbox**
- int **socket**
- int **recv_timeout**
- int **recv_bufsize**
- struct **api_msg_msg** * **write_msg**
- int **write_offset**
- u8_t **write_delayed**
- netconn_callback **callback**

Detailed Description

A **netconn** descriptor

Field Documentation

enum netconn_type netconn::type

type of the **netconn** (TCP, UDP or RAW)

enum netconn_state netconn::state

current state of the **netconn**

err_t netconn::err

the last error this **netconn** had

sys_sem_t netconn::op_completed

sem that is used to synchroneously execute functions in the core context

sys_mbox_t netconn::recvmbox

mbox where received packets are stored until they are fetched by the **netconn** application thread (can grow quite big)

sys_mbox_t netconn::acceptmbox

mbox where new connections are stored until processed by the application thread

int netconn::socket

only used for socket layer

int netconn::recv_timeout

timeout to wait for new data to be received (or connections to arrive for listening netconns)

int netconn::recv_bufsize

maximum amount of bytes queued in recvmbox

struct api_msg_msg* netconn::write_msg [read]

TCP: when data passed to netconn_write doesn't fit into the send buffer, this temporarily stores the message.

int netconn::write_offset

TCP: when data passed to netconn_write doesn't fit into the send buffer, this temporarily stores how much is already sent.

u8_t netconn::write_delayed

TCP: when data passed to netconn_write doesn't fit into the send buffer, this temporarily stores whether to wake up the original application task if data couldn't be sent in the first try.

netconn_callback netconn::callback

A callback function that is informed about events for this **netconn**

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

- lwip/src/include/lwip/api.h

netif Struct Reference

`#include <lwip/src/include/lwip/netif.h>`

Data Fields

- struct **netif** * **next**
- struct **ip_addr ip_addr**
- err_t(* **input**)(struct pbuf *p, struct **netif** *inp)
- err_t(* **output**)(struct **netif** *__netif__, struct pbuf *p, struct **ip_addr** *ipaddr)
- err_t(* **linkoutput**)(struct **netif** *__netif__, struct pbuf *p)
- void(* **status_callback**)(struct **netif** *__netif__)
- void(* **link_callback**)(struct **netif** *__netif__)
- void * **state**
- struct **dhcp** * **dhcp**
- struct **autoip** * **autoip**
- u8_t **hwaddr_len**
- u8_t **hwaddr** [NETIF_MAX_HWADDR_LEN]
- u16_t **mtu**
- u8_t **flags**
- char **name** [2]
- u8_t **num**
- u8_t **link_type**
- u32_t **link_speed**
- u32_t **ts**
- u32_t **ifinoctets**

Detailed Description

Generic data structure used for all lwIP network interfaces. The following fields should be filled in by the initialization function for the device driver: hwaddr_len, hwaddr[], mtu, flags

---

Field Documentation

struct netif* netif::next `[read]`

pointer to next in linked list

struct ip_addr netif::ip_addr `[read]`

IP address configuration in network byte order

err_t(* netif::input)(struct pbuf *p, struct netif *inp)

This function is called by the network device driver to pass a packet up the TCP/IP stack.

err_t(* netif::output)(struct netif *netif, struct pbuf *p, struct ip_addr *ipaddr)

This function is called by the IP module when it wants to send a packet on the interface. This function typically first resolves the hardware address, then sends the packet.

err_t(* netif::linkoutput)(struct netif *netif, struct pbuf *p)

This function is called by the ARP module when it wants to send a packet on the interface. This function outputs the pbuf as-is on the link medium.

void(* netif::status_callback)(struct netif *netif)

This function is called when the **netif** state is set to up or down

void(* netif::link_callback)(struct netif *netif)

This function is called when the **netif** link is set to up or down

void* netif::state

This field can be set by the device driver and could point to state information for the device.

struct dhcp* netif::dhcp `[read]`

the DHCP client state information for this **netif**

struct autoip* netif::autoip `[read]`

the AutoIP client state information for this **netif**

u8_t netif::hwaddr_len

number of bytes used in hwaddr

u8_t netif::hwaddr[NETIF_MAX_HWADDR_LEN]

link level hardware address of this interface

u16_t netif::mtu

maximum transfer unit (in bytes)

u8_t netif::flags

flags (see NETIF_FLAG_ above)

char netif::name[2]

descriptive abbreviation

u8_t netif::num

number of this interface

u8_t netif::link_type

link type (from "snmp_ifType" enum from snmp.h)

u32_t netif::link_speed

(estimate) link speed

u32_t netif::ts

timestamp at last change made (up/down)

u32_t netif::ifinoctets

counters

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

- lwip/src/include/lwip/netif.h

nse Struct Reference

Data Fields

- struct **mib_node** * **r_ptr**
- s32_t **r_id**
- u8_t **r_nl**

Detailed Description

node stack entry (old news?)

Field Documentation

struct mib_node* nse::r_ptr `[read]`

right child

s32_t nse::r_id

right child identifier

u8_t nse::r_nl

right child next level

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

- lwip/src/core/snmp/**mib_structs.c**

obj_def Struct Reference

`#include <lwip/src/include/lwip/snmp_structs.h>`

Detailed Description

object definition returned by (get_object_def)()

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

- lwip/src/include/lwip/**snmp_structs.h**

snmp_obj_id Struct Reference

`#include <lwip/src/include/lwip/snmp.h>`

Detailed Description

internal object identifier representation

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

- lwip/src/include/lwip/snmp.h

snmp_resp_header_lengths Struct Reference

`#include <lwip/src/include/lwip/snmp_msg.h>`

Detailed Description

output response message header length fields

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

- lwip/src/include/lwip/**snmp_msg.h**

snmp_trap_header_lengths Struct Reference

`#include <lwip/src/include/lwip/snmp_msg.h>`

Detailed Description

output response message header length fields

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

- lwip/src/include/lwip/**snmp_msg.h**

sswt_cb Struct Reference

Detailed Description

Struct used for **sys_sem_wait_timeout()** to tell wether the time has run out or the semaphore has really become available.

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

- lwip/src/core/**sys.c**

lwIP 1.3.0 File Documentation

lwip/src/api/api_lib.c File Reference

Functions

- struct **netconn** * **netconn_new_with_proto_and_callback** (enum netconn_type t, u8_t proto, netconn_callback callback)
- err_t **netconn_delete** (struct **netconn** *conn)
- enum netconn_type **netconn_type** (struct **netconn** *conn)
- err_t **netconn_getaddr** (struct **netconn** *conn, struct ip_addr *addr, u16_t *port, u8_t local)
- err_t **netconn_bind** (struct **netconn** *conn, struct ip_addr *addr, u16_t port)
- err_t **netconn_connect** (struct **netconn** *conn, struct ip_addr *addr, u16_t port)
- err_t **netconn_disconnect** (struct **netconn** *conn)
- err_t **netconn_listen_with_backlog** (struct **netconn** *conn, u8_t backlog)
- struct **netconn** * **netconn_accept** (struct **netconn** *conn)
- struct netbuf * **netconn_recv** (struct **netconn** *conn)
- err_t **netconn_sendto** (struct **netconn** *conn, struct netbuf *buf, struct ip_addr *addr, u16_t port)
- err_t **netconn_send** (struct **netconn** *conn, struct netbuf *buf)
- err_t **netconn_write** (struct **netconn** *conn, const void *dataptr, int size, u8_t apiflags)
- err_t **netconn_close** (struct **netconn** *conn)
- err_t **netconn_join_leave_group** (struct **netconn** *conn, struct ip_addr *multiaddr, struct ip_addr *interface, enum netconn_igmp join_or_leave)
- err_t **netconn_gethostbyname** (const char *name, struct ip_addr *addr)

Detailed Description

Sequential API External module

Function Documentation

struct netconn* netconn_accept (struct netconn * *conn*) `[read]`

Accept a new connection on a TCP listening **netconn**.

Parameters:

conn the TCP listen **netconn**

Returns:

the newly accepted **netconn** or NULL on timeout

err_t netconn_bind (struct netconn * *conn*, struct ip_addr * *addr*, u16_t *port*)

Bind a **netconn** to a specific local IP address and port. Binding one **netconn** twice might not always be checked correctly!

Parameters:

conn the **netconn** to bind
addr the local IP address to bind the **netconn** to (use IP_ADDR_ANY to bind to all addresses)
port the local port to bind the **netconn** to (not used for RAW)

Returns:
ERR_OK if bound, any other err_t on failure

err_t netconn_close (struct netconn * *conn*)

Close a TCP **netconn** (doesn't delete it).

Parameters:
conn the TCP **netconn** to close

Returns:
ERR_OK if the **netconn** was closed, any other err_t on error

err_t netconn_connect (struct netconn * *conn*, struct ip_addr * *addr*, u16_t *port*)

Connect a **netconn** to a specific remote IP address and port.

Parameters:
conn the **netconn** to connect
addr the remote IP address to connect to
port the remote port to connect to (no used for RAW)

Returns:
ERR_OK if connected, return value of tcp_/udp_/raw_connect otherwise

err_t netconn_delete (struct netconn * *conn*)

Close a **netconn** 'connection' and free its resources. UDP and RAW connection are completely closed, TCP pcbs might still be in a waitstate after this returns.

Parameters:
conn the **netconn** to delete

Returns:
ERR_OK if the connection was deleted

err_t netconn_disconnect (struct netconn * *conn*)

Disconnect a **netconn** from its current peer (only valid for UDP netconns).

Parameters:
conn the **netconn** to disconnect

Returns:
TODO: return value is not set here...

err_t netconn_getaddr (struct netconn * *conn*, struct ip_addr * *addr*, u16_t * *port*, u8_t *local*)

Get the local or remote IP address and port of a **netconn**. For RAW netconns, this returns the protocol instead of a port!

Parameters:
conn the **netconn** to query
addr a pointer to which to save the IP address
port a pointer to which to save the port (or protocol for RAW)
local 1 to get the local IP address, 0 to get the remote one

Returns:
ERR_CONN for invalid connections ERR_OK if the information was retrieved

err_t netconn_gethostbyname (const char * *name*, struct ip_addr * *addr*)

Execute a DNS query, only one IP address is returned

Parameters:

name a string representation of the DNS host name to query
addr a preallocated struct ip_addr where to store the resolved IP address

Returns:

ERR_OK: resolving succeeded ERR_MEM: memory error, try again later ERR_ARG: dns client not initialized or invalid hostname ERR_VAL: dns server response was invalid

err_t netconn_join_leave_group (struct netconn * *conn*, struct ip_addr * *multiaddr*, struct ip_addr * *interface*, enum netconn_igmp *join_or_leave*)

Join multicast groups for UDP netconns.

Parameters:

conn the UDP **netconn** for which to change multicast addresses
multiaddr IP address of the multicast group to join or leave
interface the IP address of the network interface on which to send the igmp message
join_or_leave flag whether to send a join- or leave-message

Returns:

ERR_OK if the action was taken, any err_t on error

err_t netconn_listen_with_backlog (struct netconn * *conn*, u8_t *backlog*)

Set a TCP **netconn** into listen mode

Parameters:

conn the tcp **netconn** to set to listen mode
backlog the listen backlog, only used if TCP_LISTEN_BACKLOG==1

Returns:

ERR_OK if the **netconn** was set to listen (UDP and RAW netconns don't return any error (yet?))

struct netconn* netconn_new_with_proto_and_callback (enum netconn_type *t*, u8_t *proto*, netconn_callback *callback*) `[read]`

Create a new **netconn** (of a specific type) that has a callback function. The corresponding pcb is also created.

Parameters:

t the type of 'connection' to create (

See also:

enum **netconn_type**)

Parameters:

proto the IP protocol for RAW IP pcbs
callback a function to call on status changes (RX available, TX'ed)

Returns:

a newly allocated struct **netconn** or NULL on memory error

struct netbuf* netconn_recv (struct netconn * *conn*) `[read]`

Receive data (in form of a netbuf containing a packet buffer) from a **netconn**

Parameters:

conn the **netconn** from which to receive data

Returns:

a new netbuf containing received data or NULL on memory error or timeout

err_t netconn_send (struct netconn * *conn*, struct netbuf * *buf*)

Send data over a UDP or RAW **netconn** (that is already connected).

Parameters:

conn the UDP or RAW **netconn** over which to send data
buf a netbuf containing the data to send

Returns:

ERR_OK if data was sent, any other err_t on error

err_t netconn_sendto (struct netconn * *conn*, struct netbuf * *buf*, struct ip_addr * *addr*, u16_t *port*)

Send data (in form of a netbuf) to a specific remote IP address and port. Only to be used for UDP and RAW netconns (not TCP).

Parameters:

conn the **netconn** over which to send data
buf a netbuf containing the data to send
addr the remote IP address to which to send the data
port the remote port to which to send the data

Returns:

ERR_OK if data was sent, any other err_t on error

enum netconn_type netconn_type (struct netconn * *conn*)

Get the type of a **netconn** (as enum netconn_type).

Parameters:

conn the **netconn** of which to get the type

Returns:

the netconn_type of conn

err_t netconn_write (struct netconn * *conn*, const void * *dataptr*, int *size*, u8_t *apiflags*)

Send data over a TCP **netconn**.

Parameters:

conn the TCP **netconn** over which to send data
dataptr pointer to the application buffer that contains the data to send
size size of the application data to send
apiflags combination of following flags :

- NETCONN_COPY (0x01) data will be copied into memory belonging to the stack
- NETCONN_MORE (0x02) for TCP connection, PSH flag will be set on last segment sent

Returns:

ERR_OK if data was sent, any other err_t on error

lwip/src/api/api_msg.c File Reference

Detailed Description

Sequential API Internal module

lwip/src/api/err.c File Reference

Functions

- const char * **lwip_strerr** (err_t err)

Detailed Description

Error Management module

Function Documentation

const char* lwip_strerr (err_t *err*)

Convert an lwip internal error to a string representation.

Parameters:

err an lwip internal err_t

Returns:

a string representation for err

lwip/src/api/netbuf.c File Reference

Functions

- struct netbuf * **netbuf_new** (void)
- void **netbuf_delete** (struct netbuf *buf)
- void * **netbuf_alloc** (struct netbuf *buf, u16_t size)
- void **netbuf_free** (struct netbuf *buf)
- err_t **netbuf_ref** (struct netbuf *buf, const void *dataptr, u16_t size)
- void **netbuf_chain** (struct netbuf *head, struct netbuf *tail)
- err_t **netbuf_data** (struct netbuf *buf, void **dataptr, u16_t *len)
- s8_t **netbuf_next** (struct netbuf *buf)
- void **netbuf_first** (struct netbuf *buf)

Detailed Description

Network buffer management

Function Documentation

void* netbuf_alloc (struct netbuf * *buf*, u16_t *size*)

Allocate memory for a packet buffer for a given netbuf.

Parameters:

buf the netbuf for which to allocate a packet buffer
size the size of the packet buffer to allocate

Returns:

pointer to the allocated memory NULL if no memory could be allocated

void netbuf_chain (struct netbuf * *head*, struct netbuf * *tail*)

Chain one netbuf to another (

See also:

pbuf_chain)

Parameters:

head the first netbuf
tail netbuf to chain after head

err_t netbuf_data (struct netbuf * *buf*, void ** *dataptr*, u16_t * *len*)

Get the data pointer and length of the data inside a netbuf.

Parameters:

buf netbuf to get the data from
dataptr pointer to a void pointer where to store the data pointer
len pointer to an u16_t where the length of the data is stored

Returns:

ERR_OK if the information was retreived, ERR_BUF on error.

void netbuf_delete (struct netbuf * *buf*)

Deallocate a netbuf allocated by **netbuf_new**().

Parameters:

buf pointer to a netbuf allocated by **netbuf_new**()

void netbuf_first (struct netbuf * *buf*)

Move the current data pointer of a packet buffer contained in a netbuf to the beginning of the packet. The packet buffer itself is not modified.

Parameters:

buf the netbuf to modify

void netbuf_free (struct netbuf * *buf*)

Free the packet buffer included in a netbuf

Parameters:

buf pointer to the netbuf which contains the packet buffer to free

struct netbuf* netbuf_new (void) `[read]`

Create (allocate) and initialize a new netbuf. The netbuf doesn't yet contain a packet buffer!

Returns:

a pointer to a new netbuf NULL on lack of memory

s8_t netbuf_next (struct netbuf * *buf*)

Move the current data pointer of a packet buffer contained in a netbuf to the next part. The packet buffer itself is not modified.

Parameters:

buf the netbuf to modify

Returns:

-1 if there is no next part 1 if moved to the next part but now there is no next part 0 if moved to the next part and there are still more parts

err_t netbuf_ref (struct netbuf * *buf*, const void * *dataptr*, u16_t *size*)

Let a netbuf reference existing (non-volatile) data.

Parameters:

buf netbuf which should reference the data
dataptr pointer to the data to reference
size size of the data

Returns:

ERR_OK if data is referenced ERR_MEM if data couldn't be referenced due to lack of memory

lwip/src/api/netdb.c File Reference

Data Structures

- struct **gethostbyname_r_helper**

Functions

- struct hostent * **lwip_gethostbyname** (const char *name)
- int **lwip_gethostbyname_r** (const char *name, struct hostent *ret, char *buf, size_t buflen, struct hostent **result, int *h_errnop)
- void **lwip_freeaddrinfo** (struct addrinfo *ai)
- int **lwip_getaddrinfo** (const char *nodename, const char *servname, const struct addrinfo *hints, struct addrinfo **res)

Variables

- int **h_errno**

Detailed Description

API functions for name resolving

Function Documentation

void lwip_freeaddrinfo (struct addrinfo * *ai*)

Frees one or more addrinfo structures returned by getaddrinfo(), along with any additional storage associated with those structures. If the ai_next field of the structure is not null, the entire list of structures is freed.

Parameters:

ai struct addrinfo to free

int lwip_getaddrinfo (const char * *nodename*, const char * *servname*, const struct addrinfo * *hints*, struct addrinfo ** *res*)

Translates the name of a service location (for example, a host name) and/or a service name and returns a set of socket addresses and associated information to be used in creating a socket with which to address the specified service. Memory for the result is allocated internally and must be freed by calling **lwip_freeaddrinfo()**!

Due to a limitation in dns_gethostbyname, only the first address of a host is returned. Also, service names are not supported (only port numbers)!

Parameters:

nodename descriptive name or address string of the host (may be NULL -> local address)
servname port number as string of NULL
hints structure containing input values that set socktype and protocol
res pointer to a pointer where to store the result (set to NULL on failure)

Returns:

0 on success, non-zero on failure

struct hostent* lwip_gethostbyname (const char * *name*) `[read]`

Returns an entry containing addresses of address family AF_INET for the host with name name. Due to dns_gethostbyname limitations, only one address is returned.

Parameters:

name the hostname to resolve

Returns:

an entry containing addresses of address family AF_INET for the host with name name

int lwip_gethostbyname_r (const char * *name*, struct hostent * *ret*, char * *buf*, size_t *buflen*, struct hostent ** *result*, int * *h_errnop*)

Thread-safe variant of lwip_gethostbyname: instead of using a static buffer, this function takes buffer and errno pointers as arguments and uses these for the result.

Parameters:

name the hostname to resolve
ret pre-allocated struct where to store the result
buf pre-allocated buffer where to store additional data
buflen the size of buf
result pointer to a hostent pointer that is set to ret on success and set to zero on error
h_errnop pointer to an int where to store errors (instead of modifying the global h_errno)

Returns:

0 on success, non-zero on error, additional error information is stored in *h_errnop instead of h_errno to be thread-safe

---

Variable Documentation

int h_errno

h_errno is exported in netdb.h for access by applications.

lwip/src/api/netifapi.c File Reference

Functions

- void **do_netifapi_netif_add** (struct netifapi_msg_msg *msg)
- void **do_netifapi_netif_common** (struct netifapi_msg_msg *msg)
- err_t **netifapi_netif_add** (struct **netif** ***netif**, struct ip_addr *ipaddr, struct ip_addr *netmask, struct ip_addr *gw, void *state, err_t(*init)(struct **netif** ***netif**), err_t(*input)(struct pbuf *p, struct **netif** ***netif**))
- err_t **netifapi_netif_common** (struct **netif** ***netif**, void(*voidfunc)(struct **netif** ***netif**), err_t(*errtfunc)(struct **netif** ***netif**))

Detailed Description

Network Interface Sequential API module

Function Documentation

void do_netifapi_netif_add (struct netifapi_msg_msg * *msg*)

Call **netif_add()** inside the tcpip_thread context.

void do_netifapi_netif_common (struct netifapi_msg_msg * *msg*)

Call the "errtfunc" (or the "voidfunc" if "errtfunc" is NULL) inside the tcpip_thread context.

err_t netifapi_netif_add (struct netif * *netif*, struct ip_addr * *ipaddr*, struct ip_addr * *netmask*, struct ip_addr * *gw*, void * *state*, err_t(*)(struct netif *netif) *init*, err_t(*)(struct pbuf *p, struct netif *netif) *input*)

Call **netif_add()** in a thread-safe way by running that function inside the tcpip_thread context.

Note:

for params

See also:

netif_add()

err_t netifapi_netif_common (struct netif * *netif*, void(*)(struct netif *netif) *voidfunc*, err_t(*)(struct netif *netif) *errtfunc*)

call the "errtfunc" (or the "voidfunc" if "errtfunc" is NULL) in a thread-safe way by running that function inside the tcpip_thread context.

Note:

use only for functions where there is only "netif" parameter.

lwip/src/api/sockets.c File Reference

Data Structures

- struct **lwip_socket**
- struct **lwip_select_cb**
- struct **lwip_setgetsockopt_data**

Functions

- void **lwip_socket_init** (void)
- int **lwip_listen** (int s, int backlog)
- int **lwip_select** (int maxfdp1, fd_set *readset, fd_set *writeset, fd_set *exceptset, struct timeval *timeout)
- int **lwip_shutdown** (int s, int how)

Detailed Description

Sockets BSD-Like API module

Function Documentation

int lwip_listen (int *s*, int *backlog*)

Set a socket into listen mode. The socket may not have been used for another connection previously.

Parameters:

s the socket to set to listening mode
backlog (ATTENTION: need TCP_LISTEN_BACKLOG=1)

Returns:

0 on success, non-zero on failure

int lwip_select (int *maxfdp1*, fd_set * *readset*, fd_set * *writeset*, fd_set * *exceptset*, struct timeval * *timeout*)

Processing exceptset is not yet implemented.

int lwip_shutdown (int *s*, int *how*)

Unimplemented: Close one end of a full-duplex connection. Currently, the full connection is closed.

void lwip_socket_init (void)

Initialize this module. This function has to be called before any other functions in this module!

lwip/src/api/tcpip.c File Reference

Functions

- void **tcp_timer_needed** (void)
- err_t **tcpip_input** (struct pbuf *p, struct **netif** *inp)
- err_t **tcpip_callback_with_block** (void(*f)(void *ctx), void *ctx, u8_t block)
- err_t **tcpip_apimsg** (struct **api_msg** *apimsg)
- err_t **tcpip_apimsg_lock** (struct **api_msg** *apimsg)
- err_t **tcpip_netifapi** (struct netifapi_msg *netifapimsg)
- err_t **tcpip_netifapi_lock** (struct netifapi_msg *netifapimsg)
- void **tcpip_init** (void(*initfunc)(void *), void *arg)

Variables

- sys_sem_t **lock_tcpip_core**

Detailed Description

Sequential API Main thread module

Function Documentation

void tcp_timer_needed (void)

Called from TCP_REG when registering a new PCB: the reason is to have the TCP timer only running when there are active (or time-wait) PCBs.

err_t tcpip_apimsg (struct api_msg * *apimsg*)

Call the lower part of a netconn_* function This function is then running in the thread context of tcpip_thread and has exclusive access to lwIP core code.

Parameters:

apimsg a struct containing the function to call and its parameters

Returns:

ERR_OK if the function was called, another err_t if not

err_t tcpip_apimsg_lock (struct api_msg * *apimsg*)

Call the lower part of a netconn_* function This function has exclusive access to lwIP core code by locking it before the function is called.

Parameters:

apimsg a struct containing the function to call and its parameters

Returns:

ERR_OK (only for compatibility fo **tcpip_apimsg**())

err_t tcpip_callback_with_block (void(*)(void *ctx) *f*, void * *ctx*, u8_t *block*)

Call a specific function in the thread context of tcpip_thread for easy access synchronization. A function called in that way may access lwIP core code without fearing concurrent access.

Parameters:

f the function to call
ctx parameter passed to f
block 1 to block until the request is posted, 0 to non-blocking mode

Returns:

ERR_OK if the function was called, another err_t if not

void tcpip_init (void(*)(void *) *initfunc*, void * *arg*)

Initialize this module:

- initialize all sub modules
- start the tcpip_thread

Parameters:

initfunc a function to call when tcpip_thread is running and finished initializing
arg argument to pass to initfunc

err_t tcpip_input (struct pbuf * *p*, struct netif * *inp*)

Pass a received packet to tcpip_thread for input processing

Parameters:

p the received packet, p->payload pointing to the Ethernet header or to an IP header (if **netif** doesn't got NETIF_FLAG_ETHARP flag)
inp the network interface on which the packet was received

err_t tcpip_netifapi (struct netifapi_msg * *netifapimsg*)

Much like tcpip_apimsg, but calls the lower part of a netifapi_* function.

Parameters:

netifapimsg a struct containing the function to call and its parameters

Returns:

error code given back by the function that was called

err_t tcpip_netifapi_lock (struct netifapi_msg * *netifapimsg*)

Call the lower part of a netifapi_* function This function has exclusive access to lwIP core code by locking it before the function is called.

Parameters:

netifapimsg a struct containing the function to call and its parameters

Returns:

ERR_OK (only for compatibility fo **tcpip_netifapi**())

Variable Documentation

sys_sem_t lock_tcpip_core

The global semaphore to lock the stack.

lwip/src/core/dhcp.c File Reference

Functions

- void **dhcp_coarse_tmr** ()
- void **dhcp_fine_tmr** ()
- err_t **dhcp_start** (struct **netif** *__netif__)
- void **dhcp_inform** (struct **netif** *__netif__)
- void **dhcp_arp_reply** (struct **netif** *__netif__, struct ip_addr *addr)
- err_t **dhcp_renew** (struct **netif** *__netif__)
- err_t **dhcp_release** (struct **netif** *__netif__)
- void **dhcp_stop** (struct **netif** *__netif__)

Detailed Description

Dynamic Host Configuration Protocol client

Function Documentation

void dhcp_arp_reply (struct netif * *netif*, struct ip_addr * *addr*)

Match an ARP reply with the offered IP address.

Parameters:

netif the network interface on which the reply was received
addr The IP address we received a reply from

void dhcp_coarse_tmr (void)

The DHCP timer that checks for lease renewal/rebind timeouts.

void dhcp_fine_tmr (void)

DHCP transaction timeout handling

A DHCP server is expected to respond within a short period of time. This timer checks whether an outstanding DHCP request is timed out.

void dhcp_inform (struct netif * *netif*)

Inform a DHCP server of our manual configuration.

This informs DHCP servers of our fixed IP address configuration by sending an INFORM message. It does not involve DHCP address configuration, it is just here to be nice to the network.

Parameters:

netif The lwIP network interface

err_t dhcp_release (struct netif * *netif*)

Release a DHCP lease.

Parameters:

netif network interface which must release its lease

err_t dhcp_renew (struct netif * *netif*)

Renew an existing DHCP lease at the involved DHCP server.

Parameters:

netif network interface which must renew its lease

err_t dhcp_start (struct netif * *netif*)

Start DHCP negotiation for a network interface.

If no DHCP client instance was attached to this interface, a new client is created first. If a DHCP client instance was already present, it restarts negotiation.

Parameters:

netif The lwIP network interface

Returns:

lwIP error code

- ERR_OK - No error
- ERR_MEM - Out of memory

void dhcp_stop (struct netif * *netif*)

Remove the DHCP client from the interface.

Parameters:

netif The network interface to stop DHCP on

lwip/src/core/dns.c File Reference

Data Structures

- struct **dns_hdr**
- struct **dns_query**
- struct **dns_answer**
- struct **dns_table_entry**

Functions

- void **dns_init** ()
- void **dns_setserver** (u8_t numdns, struct ip_addr *dnsserver)
- struct ip_addr **dns_getserver** (u8_t numdns)
- void **dns_tmr** (void)
- err_t **dns_gethostbyname** (const char *hostname, struct ip_addr *addr, dns_found_callback found, void *callback_arg)

Detailed Description

DNS - host name to IP address resolver.

Function Documentation

err_t dns_gethostbyname (const char * *hostname*, struct ip_addr * *addr*, dns_found_callback *found*, void * *callback_arg*)

Resolve a hostname (string) into an IP address. NON-BLOCKING callback version for use with raw API!!!

Returns immediately with one of err_t return codes:

- ERR_OK if hostname is a valid IP address string or the host name is already in the local names table.
- ERR_INPROGRESS enqueue a request to be sent to the DNS server for resolution if no errors are present.

Parameters:

hostname the hostname that is to be queried
addr pointer to a struct ip_addr where to store the address if it is already cached in the dns_table (only valid if ERR_OK is returned!)
found a callback function to be called on success, failure or timeout (only if ERR_INPROGRESS is returned!)
callback_arg argument to pass to the callback function

Returns:

a err_t return code.

struct ip_addr dns_getserver (u8_t *numdns*) `[read]`

Obtain one of the currently configured DNS server.

Parameters:

numdns the index of the DNS server

Returns:

IP address of the indexed DNS server or "ip_addr_any" if the DNS server has not been configured.

void dns_init (void)

Initialize the resolver: set up the UDP pcb and configure the default server (DNS_SERVER_ADDRESS).

void dns_setserver (u8_t *numdns*, struct ip_addr * *dnsserver*)

Initialize one of the DNS servers.

Parameters:

numdns the index of the DNS server to set must be < DNS_MAX_SERVERS
dnsserver IP address of the DNS server to set

void dns_tmr (void)

The DNS resolver client timer - handle retries and timeouts and should be called every DNS_TMR_INTERVAL milliseconds (every second by default).

lwip/src/core/init.c File Reference

Functions

- void **lwip_init** (void)

Detailed Description

Modules initialization

Function Documentation

void lwip_init (void)

Perform Sanity check of user-configurable values, and initialize all modules.

lwip/src/core/ipv4/autoip.c File Reference

Functions

- void **autoip_init** (void)
- err_t **autoip_start** (struct **netif** ***netif**)
- err_t **autoip_stop** (struct **netif** ***netif**)
- void **autoip_tmr** ()
- void **autoip_arp_reply** (struct **netif** ***netif**, struct **etharp_hdr** *hdr)

Detailed Description

AutoIP Automatic LinkLocal IP Configuration

Function Documentation

void autoip_arp_reply (struct netif * *netif*, struct etharp_hdr * *hdr*)

Handles every incoming ARP Packet, called by etharp_arp_input.

Parameters:

netif network interface to use for autoip processing
hdr Incoming ARP packet

void autoip_init (void)

Initialize this module

err_t autoip_start (struct netif * *netif*)

Start AutoIP client

Parameters:

netif network interface on which start the AutoIP client

err_t autoip_stop (struct netif * *netif*)

Stop AutoIP client

Parameters:

netif network interface on which stop the AutoIP client

void autoip_tmr (void)

Has to be called in loop every AUTOIP_TMR_INTERVAL milliseconds

lwip/src/core/ipv4/icmp.c File Reference

Functions

- void **icmp_input** (struct pbuf *p, struct **netif** *inp)
- void **icmp_dest_unreach** (struct pbuf *p, enum icmp_dur_type t)
- void **icmp_time_exceeded** (struct pbuf *p, enum icmp_te_type t)

Detailed Description

ICMP - Internet Control Message Protocol

Function Documentation

void icmp_dest_unreach (struct pbuf * *p*, enum icmp_dur_type *t*)

Send an icmp 'destination unreachable' packet, called from **ip_input()** if the transport layer protocol is unknown and from **udp_input()** if the local port is not bound.

Parameters:

p the input packet for which the 'unreachable' should be sent, p->payload pointing to the IP header
t type of the 'unreachable' packet

void icmp_input (struct pbuf * *p*, struct netif * *inp*)

Processes ICMP input packets, called from **ip_input()**.

Currently only processes icmp echo requests and sends out the echo response.

Parameters:

p the icmp echo request packet, p->payload pointing to the ip header
inp the **netif** on which this packet was received

void icmp_time_exceeded (struct pbuf * *p*, enum icmp_te_type *t*)

Send a 'time exceeded' packet, called from ip_forward() if TTL is 0.

Parameters:

p the input packet for which the 'time exceeded' should be sent, p->payload pointing to the IP header
t type of the 'time exceeded' packet

lwip/src/core/ipv4/igmp.c File Reference

Functions

- void **igmp_init** (void)
- void **igmp_dump_group_list** ()
- err_t **igmp_start** (struct **netif** ***netif**)
- err_t **igmp_stop** (struct **netif** ***netif**)
- void **igmp_report_groups** (struct **netif** ***netif**)
- struct igmp_group * **igmp_lookfor_group** (struct **netif** *ifp, struct ip_addr *addr)
- struct igmp_group * **igmp_lookup_group** (struct **netif** *ifp, struct ip_addr *addr)
- err_t **igmp_remove_group** (struct igmp_group *group)
- void **igmp_input** (struct pbuf *p, struct **netif** *inp, struct ip_addr *dest)
- err_t **igmp_joingroup** (struct ip_addr *ifaddr, struct ip_addr *groupaddr)
- err_t **igmp_leavegroup** (struct ip_addr *ifaddr, struct ip_addr *groupaddr)
- void **igmp_tmr** (void)
- void **igmp_timeout** (struct igmp_group *group)
- void **igmp_start_timer** (struct igmp_group *group, u8_t max_time)
- void **igmp_stop_timer** (struct igmp_group *group)
- void **igmp_delaying_member** (struct igmp_group *group, u8_t maxresp)
- err_t **igmp_ip_output_if** (struct pbuf *p, struct ip_addr *src, struct ip_addr *dest, u8_t ttl, u8_t proto, struct **netif** ***netif**)
- void **igmp_send** (struct igmp_group *group, u8_t type)

Detailed Description

IGMP - Internet Group Management Protocol

Function Documentation

void igmp_delaying_member (struct igmp_group * *group*, u8_t *maxresp*)

Delaying membership report for a group if necessary

Parameters:

group the igmp_group for which "delaying" membership report
maxresp query delay

void igmp_dump_group_list ()

Dump global IGMP groups list

void igmp_init (void)

Initialize the IGMP module

void igmp_input (struct pbuf * *p*, struct netif * *inp*, struct ip_addr * *dest*)

Called from **ip_input**() if a new IGMP packet is received.

Parameters:

p received igmp packet, p->payload pointing to the ip header

inp network interface on which the packet was received
dest destination ip address of the igmp packet

err_t igmp_ip_output_if (struct pbuf * *p*, struct ip_addr * *src*, struct ip_addr * *dest*, u8_t *ttl*, u8_t *proto*, struct netif * *netif*)

Sends an IP packet on a network interface. This function constructs the IP header and calculates the IP header checksum. If the source IP address is NULL, the IP address of the outgoing network interface is filled in as source address.

Parameters:

p the packet to send (p->payload points to the data, e.g. next protocol header; if dest == IP_HDRINCL, p already includes an IP header and p->payload points to that IP header)
src the source IP address to send from (if src == IP_ADDR_ANY, the IP address of the **netif** used to send is used as source address)
dest the destination IP address to send the packet to
ttl the TTL value to be set in the IP header
proto the PROTOCOL to be set in the IP header
netif the **netif** on which to send this packet

Returns:

ERR_OK if the packet was sent OK ERR_BUF if p doesn't have enough space for IP/LINK headers returns errors returned by netif->output

Todo:

should be shared with **ip.c** - ip_output_if

err_t igmp_joingroup (struct ip_addr * *ifaddr*, struct ip_addr * *groupaddr*)

Join a group on one network interface.

Parameters:

ifaddr ip address of the network interface which should join a new group
groupaddr the ip address of the group which to join

Returns:

ERR_OK if group was joined on the netif(s), an err_t otherwise

Todo:

undo any other **netif** already joined

err_t igmp_leavegroup (struct ip_addr * *ifaddr*, struct ip_addr * *groupaddr*)

Leave a group on one network interface.

Parameters:

ifaddr ip address of the network interface which should leave a group
groupaddr the ip address of the group which to leave

Returns:

ERR_OK if group was left on the netif(s), an err_t otherwise

struct igmp_group* igmp_lookfor_group (struct netif * *ifp*, struct ip_addr * *addr*) `[read]`

Search for a group in the global igmp_group_list

Parameters:

ifp the network interface for which to look
addr the group ip address to search for

Returns:

a struct igmp_group* if the group has been found, NULL if the group wasn't found.

struct igmp_group* igmp_lookup_group (struct netif * *ifp*, struct ip_addr * *addr*) `[read]`

Search for a specific igmp group and create a new one if not found-

Parameters:

ifp the network interface for which to look
addr the group ip address to search

Returns:

a struct igmp_group*, NULL on memory error.

err_t igmp_remove_group (struct igmp_group * *group*)

Remove a group in the global igmp_group_list

Parameters:

group the group to remove from the global igmp_group_list

Returns:

ERR_OK if group was removed from the list, an err_t otherwise

void igmp_report_groups (struct netif * *netif*)

Report IGMP memberships for this interface

Parameters:

netif network interface on which report IGMP memberships

void igmp_send (struct igmp_group * *group*, u8_t *type*)

Send an igmp packet to a specific group.

Parameters:

group the group to which to send the packet
type the type of igmp packet to send

err_t igmp_start (struct netif * *netif*)

Start IGMP processing on interface

Parameters:

netif network interface on which start IGMP processing

void igmp_start_timer (struct igmp_group * *group*, u8_t *max_time*)

Start a timer for an igmp group

Parameters:

group the igmp_group for which to start a timer
max_time the time in multiples of IGMP_TMR_INTERVAL (decrease with every call to **igmp_tmr**())

Todo:

Important !! this should be random 0 -> max_time. Find out how to do this

err_t igmp_stop (struct netif * *netif*)

Stop IGMP processing on interface

Parameters:

netif network interface on which stop IGMP processing

void igmp_stop_timer (struct igmp_group * *group*)

Stop a timer for an igmp_group

Parameters:

group the igmp_group for which to stop the timer

void igmp_timeout (struct igmp_group * *group*)

Called if a timeout for one group is reached. Sends a report for this group.

Parameters:

group an igmp_group for which a timeout is reached

void igmp_tmr (void)

The igmp timer function (both for NO_SYS=1 and =0) Should be called every IGMP_TMR_INTERVAL milliseconds (100 ms is default).

lwip/src/core/ipv4/inet.c File Reference

Functions

- u32_t **inet_addr** (const char *cp)
- int **inet_aton** (const char *cp, struct in_addr *addr)
- char * **inet_ntoa** (struct in_addr addr)
- u16_t **htons** (u16_t n)
- u16_t **ntohs** (u16_t n)
- u32_t **htonl** (u32_t n)
- u32_t **ntohl** (u32_t n)

Detailed Description

Functions common to all TCP/IPv4 modules, such as the byte order functions.

Function Documentation

u32_t htonl (u32_t *n*)

Convert an u32_t from host- to network byte order.

Parameters:

n u32_t in host byte order

Returns:

n in network byte order

u16_t htons (u16_t *n*)

These are reference implementations of the byte swapping functions. Again with the aim of being simple, correct and fully portable. Byte swapping is the second thing you would want to optimize. You will need to port it to your architecture and in your cc.h:

define LWIP_PLATFORM_BYTESWAP 1 define LWIP_PLATFORM_HTONS(x) <your_htons> define LWIP_PLATFORM_HTONL(x) <your_htonl>

Note **ntohs()** and **ntohl()** are merely references to the htonx counterparts. Convert an u16_t from host- to network byte order.

Parameters:

n u16_t in host byte order

Returns:

n in network byte order

u32_t inet_addr (const char * *cp*)

Ascii internet address interpretation routine. The value returned is in network order.

Parameters:

cp IP address in ascii represenation (e.g. "127.0.0.1")

Returns:

ip address in network order

int inet_aton (const char * *cp*, struct in_addr * *addr*)

Check whether "cp" is a valid ascii representation of an Internet address and convert to a binary address. Returns 1 if the address is valid, 0 if not. This replaces inet_addr, the return value from which cannot distinguish between failure and a local broadcast address.

Parameters:

cp IP address in ascii represenation (e.g. "127.0.0.1")
addr pointer to which to save the ip address in network order

Returns:

1 if cp could be converted to addr, 0 on failure

char* inet_ntoa (struct in_addr *addr*)

Convert numeric IP address into decimal dotted ASCII representation. returns ptr to static buffer; not reentrant!

Parameters:

addr ip address in network order to convert

Returns:

pointer to a global static (!) buffer that holds the ASCII represenation of addr

u32_t ntohl (u32_t *n*)

Convert an u32_t from network- to host byte order.

Parameters:

n u32_t in network byte order

Returns:

n in host byte order

u16_t ntohs (u16_t *n*)

Convert an u16_t from network- to host byte order.

Parameters:

n u16_t in network byte order

Returns:

n in host byte order

lwip/src/core/ipv4/inet_chksum.c File Reference

Functions

- u16_t **inet_chksum_pbuf** (struct pbuf *p)

Detailed Description

Incluse internet checksum functions.

Function Documentation

u16_t inet_chksum_pbuf (struct pbuf * *p*)

Calculate a checksum over a chain of pbufs (without pseudo-header, much like inet_chksum only pbufs are used).

Parameters:

p pbuf chain over that the checksum should be calculated

Returns:

checksum (as u16_t) to be saved directly in the protocol header

lwip/src/core/ipv4/ip.c File Reference

Functions

- struct **netif** * **ip_route** (struct ip_addr *dest)
- err_t **ip_input** (struct pbuf *p, struct **netif** *inp)

Detailed Description

This is the IPv4 layer implementation for incoming and outgoing IP traffic.

See also:

ip_frag.c

Function Documentation

err_t ip_input (struct pbuf * *p*, struct netif * *inp*)

This function is called by the network interface device driver when an IP packet is received. The function does the basic checks of the IP header such as packet size being at least larger than the header size etc. If the packet was not destined for us, the packet is forwarded (using ip_forward). The IP checksum is always checked.

Finally, the packet is sent to the upper layer protocol input function.

Parameters:

p the received IP packet (p->payload points to IP header)
inp the **netif** on which this packet was received

Returns:

ERR_OK if the packet was processed (could return ERR_* if it wasn't processed, but currently always returns ERR_OK)

Sends an IP packet on a network interface. This function constructs the IP header and calculates the IP header checksum. If the source IP address is NULL, the IP address of the outgoing network interface is filled in as source address. If the destination IP address is IP_HDRINCL, p is assumed to already include an IP header and p->payload points to it instead of the data.

Parameters:

p the packet to send (p->payload points to the data, e.g. next protocol header; if dest == IP_HDRINCL, p already includes an IP header and p->payload points to that IP header)
src the source IP address to send from (if src == IP_ADDR_ANY, the IP address of the **netif** used to send is used as source address)
dest the destination IP address to send the packet to
ttl the TTL value to be set in the IP header
tos the TOS value to be set in the IP header
proto the PROTOCOL to be set in the IP header
netif the **netif** on which to send this packet

Returns:

ERR_OK if the packet was sent OK ERR_BUF if p doesn't have enough space for IP/LINK headers returns errors returned by netif->output

Note:

ip_id: RFC791 "some host may be able to simply use unique identifiers independent of destination"

Simple interface to ip_output_if. It finds the outgoing network interface and calls upon ip_output_if to do the actual work.

Parameters:

p the packet to send (p->payload points to the data, e.g. next protocol header; if dest == IP_HDRINCL, p already includes an IP header and p->payload points to that IP header)
src the source IP address to send from (if src == IP_ADDR_ANY, the IP address of the **netif** used to send is used as source address)
dest the destination IP address to send the packet to
ttl the TTL value to be set in the IP header
tos the TOS value to be set in the IP header
proto the PROTOCOL to be set in the IP header

Returns:

ERR_RTE if no route is found see ip_output_if() for more return values

struct netif* ip_route (struct ip_addr * *dest*) `[read]`

Finds the appropriate network interface for a given IP address. It searches the list of network interfaces linearly. A match is found if the masked IP address of the network interface equals the masked IP address given to the function.

Parameters:

dest the destination IP address for which to find the route

Returns:

the **netif** on which to send to reach dest

lwip/src/core/ipv4/ip_addr.c File Reference

Functions

- u8_t **ip_addr_isbroadcast** (struct ip_addr *addr, struct **netif** ***netif**)

Detailed Description

This is the IPv4 address tools implementation.

Function Documentation

u8_t ip_addr_isbroadcast (struct ip_addr * *addr*, struct netif * *netif*)

Determine if an address is a broadcast address on a network interface

Parameters:

addr address to be checked
netif the network interface against which the address is checked

Returns:

returns non-zero if the address is a broadcast address

lwip/src/core/ipv4/ip_frag.c File Reference

Data Structures

- struct **ip_reass_helper**

Functions

- void **ip_reass_tmr** (void)
- struct pbuf * **ip_reass** (struct pbuf *p)
- err_t **ip_frag** (struct pbuf *p, struct **netif** ***netif**, struct ip_addr *dest)

Detailed Description

This is the IPv4 packet segmentation and reassembly implementation.

Function Documentation

err_t ip_frag (struct pbuf * *p*, struct netif * *netif*, struct ip_addr * *dest*)

Fragment an IP datagram if too large for the **netif**.

Chop the datagram in MTU sized chunks and send them in order by using a fixed size static memory buffer (PBUF_REF) or point PBUF_REFs into p (depending on IP_FRAG_USES_STATIC_BUF).

Parameters:

p ip packet to send
netif the **netif** on which to send
dest destination ip address to which to send

Returns:

ERR_OK if sent successfully, err_t otherwise

struct pbuf* ip_reass (struct pbuf * *p*) `[read]`

Reassembles incoming IP fragments into an IP datagram.

Parameters:

p points to a pbuf chain of the fragment

Returns:

NULL if reassembly is incomplete, ? otherwise

void ip_reass_tmr (void)

Reassembly timer base function for both NO_SYS == 0 and 1 (!).

Should be called every 1000 msec (defined by IP_TMR_INTERVAL).

lwip/src/core/ipv6/inet6.c File Reference

Functions

- u16_t **inet_chksum_pbuf** (struct pbuf *p)

Detailed Description

Functions common to all TCP/IPv6 modules, such as the Internet checksum and the byte order functions.

Function Documentation

u16_t inet_chksum_pbuf (struct pbuf * *p*)

Calculate a checksum over a chain of pbufs (without pseudo-header, much like inet_chksum only pbufs are used).

Parameters:

p pbuf chain over that the checksum should be calculated

Returns:

checksum (as u16_t) to be saved directly in the protocol header

lwip/src/core/mem.c File Reference

Data Structures

- struct **mem_helper**
- struct **mem**

Functions

- void * **mem_malloc** (mem_size_t size)
- void **mem_free** (void *rmem)
- void **mem_init** (void)
- void * **mem_realloc** (void *rmem, mem_size_t newsize)
- void * **mem_calloc** (mem_size_t count, mem_size_t size)

Detailed Description

Dynamic memory manager

This is a lightweight replacement for the standard C library malloc().

If you want to use the standard C library malloc() instead, define MEM_LIBC_MALLOC to 1 in your lwipopts.h

To let **mem_malloc()** use pools (prevents fragmentation and is much faster than a heap but might waste some memory), define MEM_USE_POOLS to 1, define MEM_USE_CUSTOM_POOLS to 1 and create a file "lwippools.h" that includes a list of pools like this (more pools can be added between _START and _END):

Define three pools with sizes 256, 512, and 1512 bytes LWIP_MALLOC_MEMPOOL_START LWIP_MALLOC_MEMPOOL(20, 256) LWIP_MALLOC_MEMPOOL(10, 512) LWIP_MALLOC_MEMPOOL(5, 1512) LWIP_MALLOC_MEMPOOL_END

Function Documentation

void* mem_calloc (mem_size_t *count*, mem_size_t *size*)

Contiguously allocates enough space for count objects that are size bytes of memory each and returns a pointer to the allocated memory.

The allocated memory is filled with bytes of value zero.

Parameters:

count number of objects to allocate
size size of the objects to allocate

Returns:

pointer to allocated memory / NULL pointer if there is an error

void mem_free (void * *rmem*)

Free memory previously allocated by mem_malloc. Loads the pool number and calls memp_free with that pool number to put the element back into its pool

Parameters:

rmem the memory element to free

Put a struct **mem** back on the heap

Parameters:

rmem is the data portion of a struct **mem** as returned by a previous call to **mem_malloc()**

void mem_init (void)

Zero the heap and initialize start, end and lowest-free

void * mem_malloc (mem_size_t *size*)

Allocate memory: determine the smallest pool that is big enough to contain an element of 'size' and get an element from that pool.

Parameters:

size the size in bytes of the memory needed

Returns:

a pointer to the allocated memory or NULL if the pool is empty

Adam's **mem_malloc()** plus solution for bug #17922 Allocate a block of memory with a minimum of 'size' bytes.

Parameters:

size is the minimum size of the requested block in bytes.

Returns:

pointer to allocated memory or NULL if no free memory was found.

Note that the returned value will always be aligned (as defined by MEM_ALIGNMENT).

void* mem_realloc (void * *rmem*, mem_size_t *newsize*)

In contrast to its name, mem_realloc can only shrink memory, not expand it. Since the only use (for now) is in pbuf_realloc (which also can only shrink), this shouldn't be a problem!

Parameters:

rmem pointer to memory allocated by mem_malloc the is to be shrinked
newsize required size after shrinking (needs to be smaller than or equal to the previous size)

Returns:

for compatibility reasons: is always == rmem, at the moment

lwip/src/core/memp.c File Reference

Functions

- void **memp_init** (void)
- void * **memp_malloc** (memp_t type) memp_malloc_fn(memp_t type

---

Detailed Description

Dynamic pool memory manager

lwIP has dedicated pools for many structures (**netconn**, protocol control blocks, packet buffers, ...). All these pools are managed here.

Function Documentation

void memp_init (void)

Initialize this module.

Carves out memp_memory into linked lists for each pool-type.

void* memp_malloc (memp_t *type*)

Get an element from a specific pool.

Parameters:

type the pool to get an element from

the debug version has two more parameters:

Parameters:

file file name calling this function
line number of line where this function is called

Returns:

a pointer to the allocated memory or a NULL pointer on error

lwip/src/core/netif.c File Reference

Functions

- struct **netif** * **netif_add** (struct **netif** *__netif__, struct ip_addr *ipaddr, struct ip_addr *netmask, struct ip_addr *gw, void *state, err_t(*init)(struct **netif** *__netif__), err_t(*input)(struct pbuf *p, struct **netif** *__netif__))
- void **netif_set_addr** (struct **netif** *__netif__, struct ip_addr *ipaddr, struct ip_addr *netmask, struct ip_addr *gw)
- void **netif_remove** (struct **netif** *__netif__)
- struct **netif** * **netif_find** (char *name)
- void **netif_set_ipaddr** (struct **netif** *__netif__, struct ip_addr *ipaddr)
- void **netif_set_gw** (struct **netif** *__netif__, struct ip_addr *gw)
- void **netif_set_netmask** (struct **netif** *__netif__, struct ip_addr *netmask)
- void **netif_set_default** (struct **netif** *__netif__)
- void **netif_set_up** (struct **netif** *__netif__)
- void **netif_set_down** (struct **netif** *__netif__)
- u8_t **netif_is_up** (struct **netif** *__netif__)
- void **netif_set_status_callback** (struct **netif** *__netif__, void(*status_callback)(struct **netif** *__netif__))
- void **netif_set_link_up** (struct **netif** *__netif__)
- void **netif_set_link_down** (struct **netif** *__netif__)
- u8_t **netif_is_link_up** (struct **netif** *__netif__)
- void **netif_set_link_callback** (struct **netif** *__netif__, void(*link_callback)(struct **netif** *__netif__))

Variables

- struct **netif** * **netif_list**
- struct **netif** * **netif_default**

Detailed Description

lwIP network interface abstraction

Function Documentation

struct netif* netif_add (struct netif * *netif*, struct ip_addr * *ipaddr*, struct ip_addr * *netmask*, struct ip_addr * *gw*, void * *state*, err_t(*)(struct netif *netif) *init*, err_t(*)(struct pbuf *p, struct netif *netif) *input*) `[read]`

Add a network interface to the list of lwIP netifs.

Parameters:

netif a pre-allocated **netif** structure
ipaddr IP address for the new **netif**
netmask network mask for the new **netif**
gw default gateway IP address for the new **netif**
state opaque data passed to the new **netif**
init callback function that initializes the interface
input callback function that is called to pass ingress packets up in the protocol layer stack.

Returns:

netif, or NULL if failed.

struct netif* netif_find (char * *name*) `[read]`

Find a network interface by searching for its name

Parameters:

name the name of the **netif** (like netif->name) plus concatenated number in ascii representation (e.g. 'en0')

u8_t netif_is_link_up (struct netif * *netif*)

Ask if a link is up

u8_t netif_is_up (struct netif * *netif*)

Ask if an interface is up

void netif_remove (struct netif * *netif*)

Remove a network interface from the list of lwIP netifs.

Parameters:

netif the network interface to remove

void netif_set_addr (struct netif * *netif*, struct ip_addr * *ipaddr*, struct ip_addr * *netmask*, struct ip_addr * *gw*)

Change IP address configuration for a network interface (including netmask and default gateway).

Parameters:

netif the network interface to change
ipaddr the new IP address
netmask the new netmask
gw the new default gateway

void netif_set_default (struct netif * *netif*)

Set a network interface as the default network interface (used to output all packets for which no specific route is found)

Parameters:

netif the default network interface

void netif_set_down (struct netif * *netif*)

Bring an interface down, disabling any traffic processing.

Note:

: Enabling DHCP on a down interface will make it come up once configured.

See also:

dhcp_start()

void netif_set_gw (struct netif * *netif*, struct ip_addr * *gw*)

Change the default gateway for a network interface

Parameters:

netif the network interface to change
gw the new default gateway

Note:

call **netif_set_addr()** if you also want to change ip address and netmask

void netif_set_ipaddr (struct netif * *netif*, struct ip_addr * *ipaddr*)

Change the IP address of a network interface

Parameters:

netif the network interface to change
ipaddr the new IP address

Note:

call **netif_set_addr()** if you also want to change netmask and default gateway

void netif_set_link_callback (struct netif * *netif*, void(*)(struct netif *netif) *link_callback*)

Set callback to be called when link is brought up/down

void netif_set_link_down (struct netif * *netif*)

Called by a driver when its link goes down

void netif_set_link_up (struct netif * *netif*)

Called by a driver when its link goes up

For Ethernet network interfaces, we would like to send a "gratuitous ARP"; this is an ARP packet sent by a node in order to spontaneously cause other nodes to update an entry in their ARP cache. From RFC 3220 "IP Mobility Support for IPv4" section 4.6.

void netif_set_netmask (struct netif * *netif*, struct ip_addr * *netmask*)

Change the netmask of a network interface

Parameters:

netif the network interface to change
netmask the new netmask

Note:

call **netif_set_addr()** if you also want to change ip address and default gateway

void netif_set_status_callback (struct netif * *netif*, void(*)(struct netif *netif) *status_callback*)

Set callback to be called when interface is brought up/down

void netif_set_up (struct netif * *netif*)

Bring an interface up, available for processing traffic.

Note:

: Enabling DHCP on a down interface will make it come up once configured.

See also:

dhcp_start()

For Ethernet network interfaces, we would like to send a "gratuitous ARP"; this is an ARP packet sent by a node in order to spontaneously cause other nodes to update an entry in their ARP cache. From RFC 3220 "IP Mobility Support for IPv4" section 4.6.

Variable Documentation

struct netif* netif_default

The default network interface.

struct netif* netif_list

The list of network interfaces.

lwip/src/core/pbuf.c File Reference

Functions

- struct pbuf * **pbuf_alloc** (pbuf_layer layer, u16_t length, pbuf_type type)
- void **pbuf_realloc** (struct pbuf *p, u16_t new_len)
- u8_t **pbuf_header** (struct pbuf *p, s16_t header_size_increment)
- u8_t **pbuf_free** (struct pbuf *p)
- u8_t **pbuf_clen** (struct pbuf *p)
- void **pbuf_ref** (struct pbuf *p)
- void **pbuf_cat** (struct pbuf *h, struct pbuf *t)
- void **pbuf_chain** (struct pbuf *h, struct pbuf *t)
- struct pbuf * **pbuf_dechain** (struct pbuf *p)
- err_t **pbuf_copy** (struct pbuf *p_to, struct pbuf *p_from)
- u16_t **pbuf_copy_partial** (struct pbuf *buf, void *dataptr, u16_t len, u16_t offset)

Detailed Description

Packet buffer management

Packets are built from the pbuf data structure. It supports dynamic memory allocation for packet contents or can reference externally managed packet contents both in RAM and ROM. Quick allocation for incoming packets is provided through pools with fixed sized pbufs.

A packet may span over multiple pbufs, chained as a singly linked list. This is called a "pbuf chain".

Multiple packets may be queued, also using this singly linked list. This is called a "packet queue".

So, a packet queue consists of one or more pbuf chains, each of which consist of one or more pbufs. CURRENTLY, PACKET QUEUES ARE NOT SUPPORTED!!! Use helper structs to queue multiple packets.

The differences between a pbuf chain and a packet queue are very precise but subtle.

The last pbuf of a packet has a ->tot_len field that equals the ->len field. It can be found by traversing the list. If the last pbuf of a packet has a ->next field other than NULL, more packets are on the queue.

Therefore, looping through a pbuf of a single packet, has an loop end condition (tot_len == p->len), NOT (next == NULL).

Function Documentation

struct pbuf* pbuf_alloc (pbuf_layer *layer*, u16_t *length*, pbuf_type *type*) `[read]`

Allocates a pbuf of the given type (possibly a chain for PBUF_POOL type).

The actual memory allocated for the pbuf is determined by the layer at which the pbuf is allocated and the requested size (from the size parameter).

Parameters:

layer flag to define header size
length size of the pbuf's payload
type this parameter decides how and where the pbuf should be allocated as follows:

- PBUF_RAM: buffer memory for pbuf is allocated as one large chunk. This includes protocol headers as well.
- PBUF_ROM: no buffer memory is allocated for the pbuf, even for protocol headers. Additional headers must be prepended by allocating another pbuf and chain in to the front of the ROM pbuf. It is assumed that the memory used is really similar to ROM in that it is immutable and will not be changed. Memory which is dynamic should generally not be attached to PBUF_ROM pbufs. Use PBUF_REF instead.
- PBUF_REF: no buffer memory is allocated for the pbuf, even for protocol headers. It is assumed that the pbuf is only being used in a single thread. If the pbuf gets queued, then pbuf_take should be called to copy the buffer.
- PBUF_POOL: the pbuf is allocated as a pbuf chain, with pbufs from the pbuf pool that is allocated during pbuf_init().

Returns:

the allocated pbuf. If multiple pbufs where allocated, this is the first pbuf of a pbuf chain.

void pbuf_cat (struct pbuf * *h*, struct pbuf * *t*)

Concatenate two pbufs (each may be a pbuf chain) and take over the caller's reference of the tail pbuf.

Note:

The caller MAY NOT reference the tail pbuf afterwards. Use **pbuf_chain()** for that purpose.

See also:

pbuf_chain()

void pbuf_chain (struct pbuf * *h*, struct pbuf * *t*)

Chain two pbufs (or pbuf chains) together.

The caller MUST call pbuf_free(t) once it has stopped using it. Use **pbuf_cat()** instead if you no longer use t.

Parameters:

h head pbuf (chain)
t tail pbuf (chain)

Note:

The pbufs MUST belong to the same packet.
MAY NOT be called on a packet queue.

The ->tot_len fields of all pbufs of the head chain are adjusted. The ->next field of the last pbuf of the head chain is adjusted. The ->ref field of the first pbuf of the tail chain is adjusted.

u8_t pbuf_clen (struct pbuf * *p*)

Count number of pbufs in a chain

Parameters:

p first pbuf of chain

Returns:

the number of pbufs in a chain

err_t pbuf_copy (struct pbuf * *p_to*, struct pbuf * *p_from*)

Create PBUF_RAM copies of pbufs.

Used to queue packets on behalf of the lwIP stack, such as ARP based queueing.

Note:

You MUST explicitly use p = pbuf_take(p);
Only one packet is copied, no packet queue!

Parameters:

p_to pbuf source of the copy
p_from pbuf destination of the copy

Returns:

ERR_OK if pbuf was copied ERR_ARG if one of the pbufs is NULL or p_to is not big enough to hold p_from

u16_t pbuf_copy_partial (struct pbuf * *buf*, void * *dataptr*, u16_t *len*, u16_t *offset*)

Copy (part of) the contents of a packet buffer to an application supplied buffer.

Parameters:

buf the pbuf from which to copy data
dataptr the application supplied buffer
len length of data to copy (dataptr must be big enough)
offset offset into the packet buffer from where to begin copying len bytes

struct pbuf* pbuf_dechain (struct pbuf * *p*) `[read]`

Dechains the first pbuf from its succeeding pbufs in the chain.

Makes p->tot_len field equal to p->len.

Parameters:

p pbuf to dechain

Returns:

remainder of the pbuf chain, or NULL if it was de-allocated.

Note:

May not be called on a packet queue.

u8_t pbuf_free (struct pbuf * *p*)

Dereference a pbuf chain or queue and deallocate any no-longer-used pbufs at the head of this chain or queue.

Decrements the pbuf reference count. If it reaches zero, the pbuf is deallocated.

For a pbuf chain, this is repeated for each pbuf in the chain, up to the first pbuf which has a non-zero reference count after decrementing. So, when all reference counts are one, the whole chain is free'd.

Parameters:

p The pbuf (chain) to be dereferenced.

Returns:

the number of pbufs that were de-allocated from the head of the chain.

Note:

MUST NOT be called on a packet queue (Not verified to work yet).
the reference counter of a pbuf equals the number of pointers that refer to the pbuf (or into the pbuf).

u8_t pbuf_header (struct pbuf * *p*, s16_t *header_size_increment*)

Adjusts the payload pointer to hide or reveal headers in the payload.

Adjusts the ->payload pointer so that space for a header (dis)appears in the pbuf payload.

The ->payload, ->tot_len and ->len fields are adjusted.

Parameters:

p pbuf to change the header size.

header_size_increment Number of bytes to increment header size which increases the size of the pbuf. New space is on the front. (Using a negative value decreases the header size.) If hdr_size_inc is 0, this function does nothing and returns succesful.

PBUF_ROM and PBUF_REF type buffers cannot have their sizes increased, so the call will fail. A check is made that the increase in header size does not move the payload pointer in front of the start of the buffer.

Returns:

non-zero on failure, zero on success.

void pbuf_realloc (struct pbuf * *p*, u16_t *new_len*)

Shrink a pbuf chain to a desired length.

Parameters:

p pbuf to shrink.
new_len desired new length of pbuf chain

Depending on the desired length, the first few pbufs in a chain might be skipped and left unchanged. The new last pbuf in the chain will be resized, and any remaining pbufs will be freed.

Note:

If the pbuf is ROM/REF, only the ->tot_len and ->len fields are adjusted.
May not be called on a packet queue.
Despite its name, pbuf_realloc cannot grow the size of a pbuf (chain).

void pbuf_ref (struct pbuf * *p*)

Increment the reference count of the pbuf.

Parameters:

p pbuf to increase reference counter of

lwip/src/core/raw.c File Reference

Functions

- u8_t **raw_input** (struct pbuf *p, struct **netif** *inp)
- err_t **raw_bind** (struct raw_pcb *pcb, struct ip_addr *ipaddr)
- err_t **raw_connect** (struct raw_pcb *pcb, struct ip_addr *ipaddr)
- void **raw_recv** (struct raw_pcb *pcb, u8_t(*recv)(void *arg, struct raw_pcb *upcb, struct pbuf *p, struct ip_addr *addr), void *recv_arg)
- err_t **raw_sendto** (struct raw_pcb *pcb, struct pbuf *p, struct ip_addr *ipaddr)
- err_t **raw_send** (struct raw_pcb *pcb, struct pbuf *p)
- void **raw_remove** (struct raw_pcb *pcb)
- struct raw_pcb * **raw_new** (u8_t proto)

Detailed Description

Implementation of raw protocol PCBs for low-level handling of different types of protocols besides (or overriding) those already available in lwIP.

Function Documentation

err_t raw_bind (struct raw_pcb * *pcb*, struct ip_addr * *ipaddr*)

Bind a RAW PCB.

Parameters:

pcb RAW PCB to be bound with a local address ipaddr.
ipaddr local IP address to bind with. Use IP_ADDR_ANY to bind to all local interfaces.

Returns:

lwIP error code.

- ERR_OK. Successful. No error occured.
- ERR_USE. The specified IP address is already bound to by another RAW PCB.

See also:

raw_disconnect()

err_t raw_connect (struct raw_pcb * *pcb*, struct ip_addr * *ipaddr*)

Connect an RAW PCB. This function is required by upper layers of lwip. Using the raw api you could use **raw_sendto()** instead

This will associate the RAW PCB with the remote address.

Parameters:

pcb RAW PCB to be connected with remote address ipaddr and port.
ipaddr remote IP address to connect with.

Returns:

lwIP error code

See also:

raw_disconnect() and **raw_sendto()**

u8_t raw_input (struct pbuf * *p*, struct netif * *inp*)

Determine if in incoming IP packet is covered by a RAW PCB and if so, pass it to a user-provided receive callback function.

Given an incoming IP datagram (as a chain of pbufs) this function finds a corresponding RAW PCB and calls the corresponding receive callback function.

Parameters:

p pbuf to be demultiplexed to a RAW PCB.
inp network interface on which the datagram was received.

Returns:

- 1 if the packet has been eaten by a RAW PCB receive callback function. The caller MAY NOT not reference the packet any longer, and MAY NOT call **pbuf_free**().
- 0 if packet is not eaten (pbuf is still referenced by the caller).

struct raw_pcb* raw_new (u8_t *proto*) `[read]`

Create a RAW PCB.

Returns:

The RAW PCB which was created. NULL if the PCB data structure could not be allocated.

Parameters:

proto the protocol number of the IPs payload (e.g. IP_PROTO_ICMP)

See also:

raw_remove()

void raw_recv (struct raw_pcb * *pcb*, u8_t(*)(void *arg, struct raw_pcb *upcb, struct pbuf *p, struct ip_addr *addr) *recv*, void * *recv_arg*)

Set the callback function for received packets that match the raw PCB's protocol and binding.

The callback function MUST either

- eat the packet by calling **pbuf_free**() and returning non-zero. The packet will not be passed to other raw PCBs or other protocol layers.
- not free the packet, and return zero. The packet will be matched against further PCBs and/or forwarded to another protocol layers.

Returns:

non-zero if the packet was free()d, zero if the packet remains available for others.

void raw_remove (struct raw_pcb * *pcb*)

Remove an RAW PCB.

Parameters:

pcb RAW PCB to be removed. The PCB is removed from the list of RAW PCB's and the data structure is freed from memory.

See also:

raw_new()

err_t raw_send (struct raw_pcb * *pcb*, struct pbuf * *p*)

Send the raw IP packet to the address given by **raw_connect**()

Parameters:

pcb the raw pcb which to send
p the IP payload to send

err_t raw_sendto (struct raw_pcb * *pcb*, struct pbuf * *p*, struct ip_addr * *ipaddr*)

Send the raw IP packet to the given address. Note that actually you cannot modify the IP headers (this is inconsistent with the receive callback where you actually get the IP headers), you can only specify the IP payload here. It requires some more changes in lwIP. (there will be a **raw_send()** function then.)

Parameters:

pcb the raw pcb which to send
p the IP payload to send
ipaddr the destination address of the IP packet

lwip/src/core/snmp/asn1_dec.c File Reference

Functions

- err_t **snmp_asn1_dec_type** (struct pbuf *p, u16_t ofs, u8_t *type)
- err_t **snmp_asn1_dec_length** (struct pbuf *p, u16_t ofs, u8_t *octets_used, u16_t *length)
- err_t **snmp_asn1_dec_u32t** (struct pbuf *p, u16_t ofs, u16_t len, u32_t *value)
- err_t **snmp_asn1_dec_s32t** (struct pbuf *p, u16_t ofs, u16_t len, s32_t *value)
- err_t **snmp_asn1_dec_oid** (struct pbuf *p, u16_t ofs, u16_t len, struct **snmp_obj_id** *oid)
- err_t **snmp_asn1_dec_raw** (struct pbuf *p, u16_t ofs, u16_t len, u16_t raw_len, u8_t *raw)

Detailed Description

Abstract Syntax Notation One (ISO 8824, 8825) decoding

Todo:

not optimised (yet), favor correctness over speed, favor speed over size

---

Function Documentation

err_t snmp_asn1_dec_length (struct pbuf * *p*, u16_t *ofs*, u8_t * *octets_used*, u16_t * *length*)

Decodes length field from incoming pbuf chain into host length.

Parameters:

p points to a pbuf holding an ASN1 coded length
ofs points to the offset within the pbuf chain of the ASN1 coded length
octets_used returns number of octets used by the length code
length return host order length, upto 64k

Returns:

ERR_OK if successfull, ERR_ARG if we can't (or won't) decode

Todo:

: do we need to accept inefficient codings with many leading zero's?

err_t snmp_asn1_dec_oid (struct pbuf * *p*, u16_t *ofs*, u16_t *len*, struct snmp_obj_id * *oid*)

Decodes object identifier from incoming message into array of s32_t.

Parameters:

p points to a pbuf holding an ASN1 coded object identifier
ofs points to the offset within the pbuf chain of the ASN1 coded object identifier
len length of the coded object identifier
oid return object identifier struct

Returns:

ERR_OK if successfull, ERR_ARG if we can't (or won't) decode

err_t snmp_asn1_dec_raw (struct pbuf * *p*, u16_t *ofs*, u16_t *len*, u16_t *raw_len*, u8_t * *raw*)

Decodes (copies) raw data (ip-addresses, octet strings, opaque encoding) from incoming message into array.

Parameters:

p points to a pbuf holding an ASN1 coded raw data
ofs points to the offset within the pbuf chain of the ASN1 coded raw data
len length of the coded raw data (zero is valid, e.g. empty string!)
raw_len length of the raw return value
raw return raw bytes

Returns:

ERR_OK if successfull, ERR_ARG if we can't (or won't) decode

err_t snmp_asn1_dec_s32t (struct pbuf * *p*, u16_t *ofs*, u16_t *len*, s32_t * *value*)

Decodes integer into s32_t.

Parameters:

p points to a pbuf holding an ASN1 coded integer
ofs points to the offset within the pbuf chain of the ASN1 coded integer
len length of the coded integer field
value return host order integer

Returns:

ERR_OK if successfull, ERR_ARG if we can't (or won't) decode

Note:

ASN coded integers are _always_ signed!

err_t snmp_asn1_dec_type (struct pbuf * *p*, u16_t *ofs*, u8_t * *type*)

Retrieves type field from incoming pbuf chain.

Parameters:

p points to a pbuf holding an ASN1 coded type field
ofs points to the offset within the pbuf chain of the ASN1 coded type field
type return ASN1 type

Returns:

ERR_OK if successfull, ERR_ARG if we can't (or won't) decode

err_t snmp_asn1_dec_u32t (struct pbuf * *p*, u16_t *ofs*, u16_t *len*, u32_t * *value*)

Decodes positive integer (counter, gauge, timeticks) into u32_t.

Parameters:

p points to a pbuf holding an ASN1 coded integer
ofs points to the offset within the pbuf chain of the ASN1 coded integer
len length of the coded integer field
value return host order integer

Returns:

ERR_OK if successfull, ERR_ARG if we can't (or won't) decode

Note:

ASN coded integers are _always_ signed. E.g. +0xFFFF is coded as 0x00,0xFF,0xFF. Note the leading sign octet. A positive value of 0xFFFFFFFF is preceded with 0x00 and the length is 5 octets!!

lwip/src/core/snmp/asn1_enc.c File Reference

Functions

- void **snmp_asn1_enc_length_cnt** (u16_t length, u8_t *octets_needed)
- void **snmp_asn1_enc_u32t_cnt** (u32_t value, u16_t *octets_needed)
- void **snmp_asn1_enc_s32t_cnt** (s32_t value, u16_t *octets_needed)
- void **snmp_asn1_enc_oid_cnt** (u8_t ident_len, s32_t *ident, u16_t *octets_needed)
- err_t **snmp_asn1_enc_type** (struct pbuf *p, u16_t ofs, u8_t type)
- err_t **snmp_asn1_enc_length** (struct pbuf *p, u16_t ofs, u16_t length)
- err_t **snmp_asn1_enc_u32t** (struct pbuf *p, u16_t ofs, u8_t octets_needed, u32_t value)
- err_t **snmp_asn1_enc_s32t** (struct pbuf *p, u16_t ofs, u8_t octets_needed, s32_t value)
- err_t **snmp_asn1_enc_oid** (struct pbuf *p, u16_t ofs, u8_t ident_len, s32_t *ident)
- err_t **snmp_asn1_enc_raw** (struct pbuf *p, u16_t ofs, u8_t raw_len, u8_t *raw)

Detailed Description

Abstract Syntax Notation One (ISO 8824, 8825) encoding

Todo:

not optimised (yet), favor correctness over speed, favor speed over size

---

Function Documentation

err_t snmp_asn1_enc_length (struct pbuf * *p*, u16_t *ofs*, u16_t *length*)

Encodes host order length field into a pbuf chained ASN1 msg.

Parameters:

p points to output pbuf to encode length into
ofs points to the offset within the pbuf chain
length is the host order length to be encoded

Returns:

ERR_OK if successfull, ERR_ARG if we can't (or won't) encode

void snmp_asn1_enc_length_cnt (u16_t *length*, u8_t * *octets_needed*)

Returns octet count for length.

Parameters:

length
octets_needed points to the return value

err_t snmp_asn1_enc_oid (struct pbuf * *p*, u16_t *ofs*, u8_t *ident_len*, s32_t * *ident*)

Encodes object identifier into a pbuf chained ASN1 msg.

Parameters:

p points to output pbuf to encode oid into
ofs points to the offset within the pbuf chain
ident_len object identifier array length
ident points to object identifier array

Returns:

ERR_OK if successfull, ERR_ARG if we can't (or won't) encode

void snmp_asn1_enc_oid_cnt (u8_t *ident_len*, s32_t * *ident*, u16_t * *octets_needed*)

Returns octet count for an object identifier.

Parameters:

ident_len object identifier array length
ident points to object identifier array
octets_needed points to the return value

err_t snmp_asn1_enc_raw (struct pbuf * *p*, u16_t *ofs*, u8_t *raw_len*, u8_t * *raw*)

Encodes raw data (octet string, opaque) into a pbuf chained ASN1 msg.

Parameters:

p points to output pbuf to encode raw data into
ofs points to the offset within the pbuf chain
raw_len raw data length
raw points raw data

Returns:

ERR_OK if successfull, ERR_ARG if we can't (or won't) encode

err_t snmp_asn1_enc_s32t (struct pbuf * *p*, u16_t *ofs*, u8_t *octets_needed*, s32_t *value*)

Encodes s32_t integer into a pbuf chained ASN1 msg.

Parameters:

p points to output pbuf to encode value into
ofs points to the offset within the pbuf chain
octets_needed encoding length (from **snmp_asn1_enc_s32t_cnt**())
value is the host order s32_t value to be encoded

Returns:

ERR_OK if successfull, ERR_ARG if we can't (or won't) encode

See also:

snmp_asn1_enc_s32t_cnt()

void snmp_asn1_enc_s32t_cnt (s32_t *value*, u16_t * *octets_needed*)

Returns octet count for an s32_t.

Parameters:

value
octets_needed points to the return value

Note:

ASN coded integers are _always_ signed.

err_t snmp_asn1_enc_type (struct pbuf * *p*, u16_t *ofs*, u8_t *type*)

Encodes ASN type field into a pbuf chained ASN1 msg.

Parameters:

p points to output pbuf to encode value into
ofs points to the offset within the pbuf chain
type input ASN1 type

Returns:

ERR_OK if successfull, ERR_ARG if we can't (or won't) encode

err_t snmp_asn1_enc_u32t (struct pbuf * *p*, u16_t *ofs*, u8_t *octets_needed*, u32_t *value*)

Encodes u32_t (counter, gauge, timeticks) into a pbuf chained ASN1 msg.

Parameters:

p points to output pbuf to encode value into
ofs points to the offset within the pbuf chain
octets_needed encoding length (from **snmp_asn1_enc_u32t_cnt**())
value is the host order u32_t value to be encoded

Returns:

ERR_OK if successfull, ERR_ARG if we can't (or won't) encode

See also:

snmp_asn1_enc_u32t_cnt()

void snmp_asn1_enc_u32t_cnt (u32_t *value*, u16_t * *octets_needed*)

Returns octet count for an u32_t.

Parameters:

value
octets_needed points to the return value

Note:

ASN coded integers are _always_ signed. E.g. +0xFFFF is coded as 0x00,0xFF,0xFF. Note the leading sign octet. A positive value of 0xFFFFFFFF is preceded with 0x00 and the length is 5 octets!!

lwip/src/core/snmp/mib2.c File Reference

Functions

- void **ocstrncpy** (u8_t *dst, u8_t *src, u8_t n)
- void **objectidncpy** (s32_t *dst, s32_t *src, u8_t n)
- void **snmp_set_sysdesr** (u8_t *str, u8_t *len)
- void **snmp_set_sysobjid** (struct **snmp_obj_id** *oid)
- void **snmp_inc_sysuptime** (void)
- void **snmp_set_syscontact** (u8_t *ocstr, u8_t *ocstrlen)
- void **snmp_set_sysname** (u8_t *ocstr, u8_t *ocstrlen)
- void **snmp_set_syslocation** (u8_t *ocstr, u8_t *ocstrlen)
- void **snmp_insert_arpidx_tree** (struct **netif** *ni, struct ip_addr *ip)
- void **snmp_delete_arpidx_tree** (struct **netif** *ni, struct ip_addr *ip)
- void **snmp_insert_ipaddridx_tree** (struct **netif** *ni)
- void **snmp_delete_ipaddridx_tree** (struct **netif** *ni)
- void **snmp_insert_iprteidx_tree** (u8_t dflt, struct **netif** *ni)
- void **snmp_delete_iprteidx_tree** (u8_t dflt, struct **netif** *ni)
- void **snmp_insert_udpidx_tree** (struct udp_pcb *pcb)
- void **snmp_delete_udpidx_tree** (struct udp_pcb *pcb)
- void **noleafs_get_object_def** (u8_t ident_len, s32_t *ident, struct **obj_def** *od)

Variables

- struct **mib_list_rootnode udp_root**
- struct **mib_list_rootnode tcpconntree_root**
- struct **mib_list_rootnode ipntomtree_root**
- struct **mib_list_rootnode iprtetree_root**
- struct **mib_list_rootnode ipaddrtree_root**
- struct **mib_list_rootnode arptree_root**
- struct **mib_list_rootnode iflist_root**
- struct **mib_array_node internet**

Detailed Description

Management Information Base II (RFC1213) objects and functions.

Note:

the object identifiers for this MIB-2 and private MIB tree must be kept in sorted ascending order. This to ensure correct getnext operation.

Function Documentation

void noleafs_get_object_def (u8_t *ident_len*, s32_t * *ident*, struct obj_def * *od*)

dummy function pointers for non-leaf MIB nodes from **mib2.c**

void objectidncpy (s32_t * *dst*, s32_t * *src*, u8_t *n*)

Copy object identifier (s32_t) array.

Parameters:

dst points to destination
src points to source
n number of sub identifiers to copy.

void ocstrncpy (u8_t * *dst*, u8_t * *src*, u8_t *n*)

Copy octet string.

Parameters:

dst points to destination
src points to source
n number of octets to copy.

void snmp_delete_arpidx_tree (struct netif * *ni*, struct ip_addr * *ip*)

Removes ARP table indexes (.xIfIndex.xNetAddress) from arp table index trees.

void snmp_delete_ipaddridx_tree (struct netif * *ni*)

Removes ipAddrTable indexes (.ipAdEntAddr) from index tree.

void snmp_delete_iprteidx_tree (u8_t *dflt*, struct netif * *ni*)

Removes ipRouteTable indexes (.ipRouteDest) from index tree.

Parameters:

dflt non-zero for the default rte, zero for network rte
ni points to network interface for this rte or NULL for default route to be removed.

void snmp_delete_udpidx_tree (struct udp_pcb * *pcb*)

Removes udpTable indexes (.udpLocalAddress.udpLocalPort) from index tree.

void snmp_inc_sysuptime (void)

Must be called at regular 10 msec interval from a timer interrupt or signal handler depending on your runtime environment.

void snmp_insert_arpidx_tree (struct netif * *ni*, struct ip_addr * *ip*)

Inserts ARP table indexes (.xIfIndex.xNetAddress) into arp table index trees (both atTable and ipNetToMediaTable).

void snmp_insert_ipaddridx_tree (struct netif * *ni*)

Inserts ipAddrTable indexes (.ipAdEntAddr) into index tree.

void snmp_insert_iprteidx_tree (u8_t *dflt*, struct netif * *ni*)

Inserts ipRouteTable indexes (.ipRouteDest) into index tree.

Parameters:

dflt non-zero for the default rte, zero for network rte
ni points to network interface for this rte

Todo:

record sysuptime for _this_ route when it is installed (needed for ipRouteAge) in the **netif**.

void snmp_insert_udpidx_tree (struct udp_pcb * *pcb*)

Inserts udpTable indexes (.udpLocalAddress.udpLocalPort) into index tree.

void snmp_set_syscontact (u8_t * *ocstr*, u8_t * *ocstrlen*)

Initializes sysContact pointers, e.g. ptrs to non-volatile memory external to lwIP.

Parameters:

ocstr if non-NULL then copy str pointer
ocstrlen points to string length, excluding zero terminator

void snmp_set_sysdesr (u8_t * *str*, u8_t * *len*)

Initializes sysDescr pointers.

Parameters:

str if non-NULL then copy str pointer
len points to string length, excluding zero terminator

void snmp_set_syslocation (u8_t * *ocstr*, u8_t * *ocstrlen*)

Initializes sysLocation pointers, e.g. ptrs to non-volatile memory external to lwIP.

Parameters:

ocstr if non-NULL then copy str pointer
ocstrlen points to string length, excluding zero terminator

void snmp_set_sysname (u8_t * *ocstr*, u8_t * *ocstrlen*)

Initializes sysName pointers, e.g. ptrs to non-volatile memory external to lwIP.

Parameters:

ocstr if non-NULL then copy str pointer
ocstrlen points to string length, excluding zero terminator

void snmp_set_sysobjid (struct snmp_obj_id * *oid*)

Initializes sysObjectID value.

Parameters:

oid points to stuct **snmp_obj_id** to copy

Variable Documentation

struct mib_list_rootnode arptree_root

```
Initial value: {
  &noleafs_get_object_def,
  &noleafs_get_value,
  &noleafs_set_test,
  &noleafs_set_value,
  MIB_NODE_LR,
  0,
  NULL,
  NULL,
  0
}
```

index root node for atTable

struct mib_list_rootnode iflist_root

```
Initial value: {
  &ifentry_get_object_def,
  &ifentry_get_value,
#if SNMP_SAFE_REQUESTS
  &noleafs_set_test,
  &noleafs_set_value,
#else
  &ifentry_set_test,
  &ifentry_set_value,
#endif
  MIB_NODE_LR,
  0,
  NULL,
  NULL,
  0
}
```

index root node for ifTable

struct mib_array_node internet [read]

```
Initial value: {
  &noleafs_get_object_def,
  &noleafs_get_value,
  &noleafs_set_test,
  &noleafs_set_value,
  MIB_NODE_AR,
  2,
  internet_ids,
  internet_nodes
}
```

export MIB tree from **mib2.c**

struct mib_list_rootnode ipaddrtree_root

```
Initial value: {
  &noleafs_get_object_def,
  &noleafs_get_value,
  &noleafs_set_test,
  &noleafs_set_value,
  MIB_NODE_LR,
  0,
  NULL,
  NULL,
  0
}
```

index root node for ipAddrTable

struct mib_list_rootnode ipntomtree_root

```
Initial value: {
  &noleafs_get_object_def,
  &noleafs_get_value,
  &noleafs_set_test,
  &noleafs_set_value,
  MIB_NODE_LR,
  0,
  NULL,
  NULL,
  0
}
```

index root node for ipNetToMediaTable

struct mib_list_rootnode iprtetree_root

```
Initial value: {
  &noleafs_get_object_def,
```

```
  &noleafs_get_value,
  &noleafs_set_test,
  &noleafs_set_value,
  MIB_NODE_LR,
  0,
  NULL,
  NULL,
  0
}
```

index root node for ipRouteTable

struct mib_list_rootnode tcpconntree_root

```
Initial value: {
  &noleafs_get_object_def,
  &noleafs_get_value,
  &noleafs_set_test,
  &noleafs_set_value,
  MIB_NODE_LR,
  0,
  NULL,
  NULL,
  0
}
```

index root node for tcpConnTable

struct mib_list_rootnode udp_root

```
Initial value: {
  &noleafs_get_object_def,
  &noleafs_get_value,
  &noleafs_set_test,
  &noleafs_set_value,
  MIB_NODE_LR,
  0,
  NULL,
  NULL,
  0
}
```

index root node for udpTable

lwip/src/core/snmp/mib_structs.c File Reference

Data Structures

- struct **nse**

Functions

- void **snmp_ifindextonetif** (s32_t ifindex, struct **netif** ****netif**)
- void **snmp_netiftoifindex** (struct **netif** ***netif**, s32_t *ifidx)
- void **snmp_oidtoip** (s32_t *ident, struct ip_addr *ip)
- void **snmp_iptooid** (struct ip_addr *ip, s32_t *ident)
- s8_t **snmp_mib_node_insert** (struct **mib_list_rootnode** *rn, s32_t objid, struct mib_list_node **insn)
- s8_t **snmp_mib_node_find** (struct **mib_list_rootnode** *rn, s32_t objid, struct mib_list_node **fn)
- struct **mib_list_rootnode** * **snmp_mib_node_delete** (struct **mib_list_rootnode** *rn, struct mib_list_node *n)
- struct **mib_node** * **snmp_search_tree** (struct **mib_node** *node, u8_t ident_len, s32_t *ident, struct snmp_name_ptr *np)
- struct **mib_node** * **snmp_expand_tree** (struct **mib_node** *node, u8_t ident_len, s32_t *ident, struct **snmp_obj_id** *oidret)
- u8_t **snmp_iso_prefix_tst** (u8_t ident_len, s32_t *ident)
- u8_t **snmp_iso_prefix_expand** (u8_t ident_len, s32_t *ident, struct **snmp_obj_id** *oidret)

Variables

- const s32_t **prefix** [4] = {1, 3, 6, 1}

Detailed Description

MIB tree access/construction functions.

Function Documentation

struct mib_node* snmp_expand_tree (struct mib_node * *node*, u8_t *ident_len*, s32_t * *ident*, struct snmp_obj_id * *oidret*) `[read]`

Tree expansion.

void snmp_ifindextonetif (s32_t *ifindex*, struct netif ** *netif*)

Conversion from ifIndex to lwIP **netif**

Parameters:

ifindex is a s32_t object sub-identifier
netif points to returned **netif** struct pointer

void snmp_iptooid (struct ip_addr * *ip*, s32_t * *ident*)

Conversion from lwIP ip_addr to oid

Parameters:

ip points to input struct
ident points to s32_t ident[4] output

u8_t snmp_iso_prefix_expand (u8_t *ident_len*, s32_t * *ident*, struct snmp_obj_id * *oidret*)

Expands object identifier to the **iso.org.dod.internet** prefix for use in getnext operation.

Parameters:

ident_len the length of the supplied object identifier
ident points to the array of sub identifiers
oidret points to returned expanded object identifier

Returns:

1 if it matches, 0 otherwise

Note:

ident_len 0 is allowed, expanding to the first known object id!!

u8_t snmp_iso_prefix_tst (u8_t *ident_len*, s32_t * *ident*)

Test object identifier for the **iso.org.dod.internet** prefix.

Parameters:

ident_len the length of the supplied object identifier
ident points to the array of sub identifiers

Returns:

1 if it matches, 0 otherwise

struct mib_list_rootnode* snmp_mib_node_delete (struct mib_list_rootnode * *rn*, struct mib_list_node * *n*) `[read]`

Removes node from idx list if it has a single child left.

Parameters:

rn points to the root node
n points to the node to delete

Returns:

the nptr to be freed by caller

s8_t snmp_mib_node_find (struct mib_list_rootnode * *rn*, s32_t *objid*, struct mib_list_node ** *fn*)

Finds node in idx list and returns deletion mark.

Parameters:

rn points to the root node
objid is the object sub identifier
fn returns pointer to found node

Returns:

0 if not found, 1 if deletable, 2 can't delete (2 or more children), 3 not a list_node

s8_t snmp_mib_node_insert (struct mib_list_rootnode * *rn*, s32_t *objid*, struct mib_list_node ** *insn*)

Inserts node in idx list in a sorted (ascending order) fashion and allocates the node if needed.

Parameters:

rn points to the root node
objid is the object sub identifier
insn points to a pointer to the inserted node used for constructing the tree.

Returns:

-1 if failed, 1 if inserted, 2 if present.

void snmp_netiftoifindex (struct netif * *netif*, s32_t * *ifidx*)

Conversion from lwIP **netif** to ifIndex

Parameters:

netif points to a **netif** struct
ifidx points to s32_t object sub-identifier

void snmp_oidtoip (s32_t * *ident*, struct ip_addr * *ip*)

Conversion from oid to lwIP ip_addr

Parameters:

ident points to s32_t ident[4] input
ip points to output struct

struct mib_node* snmp_search_tree (struct mib_node * *node*, u8_t *ident_len*, s32_t * *ident*, struct snmp_name_ptr * *np*) `[read]`

Searches tree for the supplied (scalar?) object identifier.

Parameters:

node points to the root of the tree ('.internet')
ident_len the length of the supplied object identifier
ident points to the array of sub identifiers
np points to the found object instance (rerurn)

Returns:

pointer to the requested parent (!) node if success, NULL otherwise

---

Variable Documentation

const s32_t prefix[4] = {1, 3, 6, 1}

.**iso.org.dod.internet** address prefix,

See also:

snmp_iso_*()

lwip/src/core/snmp/msg_in.c File Reference

Functions

- void **snmp_init** (void)
- void **snmp_msg_event** (u8_t request_id)
- struct snmp_varbind * **snmp_varbind_alloc** (struct **snmp_obj_id** *oid, u8_t type, u8_t len)

Variables

- const s32_t **snmp_version** = 0
- const char **snmp_publiccommunity** [7] = "public"

Detailed Description

SNMP input message processing (RFC1157).

Function Documentation

void snmp_init (void)

Starts SNMP Agent. Allocates UDP pcb and binds it to IP_ADDR_ANY port 161.

void snmp_msg_event (u8_t *request_id*)

Handle one internal or external event. Called for one async event. (recv external/private answer)

Parameters:

request_id identifies requests from 0 to (SNMP_CONCURRENT_REQUESTS-1)

struct snmp_varbind* snmp_varbind_alloc (struct snmp_obj_id * *oid*, u8_t *type*, u8_t *len*) `[read]`

Varbind-list functions.

Variable Documentation

const char snmp_publiccommunity[7] = "public"

default SNMP community string

const s32_t snmp_version = 0

SNMP v1 == 0

lwip/src/core/snmp/msg_out.c File Reference

Functions

- void **snmp_trap_dst_enable** (u8_t dst_idx, u8_t enable)
- void **snmp_trap_dst_ip_set** (u8_t dst_idx, struct ip_addr *dst)
- err_t **snmp_send_response** (struct snmp_msg_pstat *m_stat)
- err_t **snmp_send_trap** (s8_t generic_trap, struct **snmp_obj_id** *eoid, s32_t specific_trap)

Variables

- struct snmp_msg_trap **trap_msg**

Detailed Description

SNMP output message processing (RFC1157).

Output responses and traps are build in two passes:

Pass 0: iterate over the output message backwards to determine encoding lengths Pass 1: the actual forward encoding of internal form into ASN1

The single-pass encoding method described by Comer & Stevens requires extra buffer space and copying for reversal of the packet. The buffer requirement can be prohibitively large for big payloads (>= 484) therefore we use the two encoding passes.

Function Documentation

err_t snmp_send_response (struct snmp_msg_pstat * *m_stat*)

Sends a 'getresponse' message to the request originator.

Parameters:

m_stat points to the current message request state source

Returns:

ERR_OK when success, ERR_MEM if we're out of memory

Note:

the caller is responsible for filling in outvb in the m_stat and provide error-status and index (except for tooBig errors) ...

Todo:

do we need separate rx and tx pcbs for threaded case?

connect to the originating source

Todo:

release some memory, retry and return tooBig? tooMuchHassle?

disassociate remote address and port with this pcb

err_t snmp_send_trap (s8_t *generic_trap*, struct snmp_obj_id * *eoid*, s32_t *specific_trap*)

Sends an generic or enterprise specific trap message.

Parameters:

generic_trap is the trap code
eoid points to enterprise object identifier
specific_trap used for enterprise traps when generic_trap == 6

Returns:

ERR_OK when success, ERR_MEM if we're out of memory

Note:

the caller is responsible for filling in outvb in the trap_msg
the use of the enterpise identifier field is per RFC1215. Use .iso.org.dod.internet.mgmt.mib-2.snmp for generic traps and .iso.org.dod.internet.private.enterprises.yourenterprise (sysObjectID) for specific traps.

connect to the TRAP destination

disassociate remote address and port with this pcb

void snmp_trap_dst_enable (u8_t *dst_idx*, u8_t *enable*)

Sets enable switch for this trap destination.

Parameters:

dst_idx index in 0 .. SNMP_TRAP_DESTINATIONS-1
enable switch if 0 destination is disabled >0 enabled.

void snmp_trap_dst_ip_set (u8_t *dst_idx*, struct ip_addr * *dst*)

Sets IPv4 address for this trap destination.

Parameters:

dst_idx index in 0 .. SNMP_TRAP_DESTINATIONS-1
dst IPv4 address in host order.

Variable Documentation

struct snmp_msg_trap trap_msg

TRAP message structure

lwip/src/core/stats.c File Reference

Detailed Description

Statistics module

lwip/src/core/sys.c File Reference

Data Structures

- struct **sswt_cb**

Functions

- void **sys_mbox_fetch** (sys_mbox_t mbox, void **msg)
- void **sys_sem_wait** (sys_sem_t sem)
- void **sys_timeout** (u32_t msecs, sys_timeout_handler h, void *arg)
- void **sys_untimeout** (sys_timeout_handler h, void *arg)
- int **sys_sem_wait_timeout** (sys_sem_t sem, u32_t timeout)
- void **sys_msleep** (u32_t ms)

Detailed Description

lwIP Operating System abstraction

Function Documentation

void sys_mbox_fetch (sys_mbox_t *mbox*, void ** *msg*)

Wait (forever) for a message to arrive in an mbox. While waiting, timeouts (for this thread) are processed.

Parameters:

mbox the mbox to fetch the message from
msg the place to store the message

void sys_msleep (u32_t *ms*)

Sleep for some ms. Timeouts are processed while sleeping.

Parameters:

ms number of milliseconds to sleep

void sys_sem_wait (sys_sem_t *sem*)

Wait (forever) for a semaphore to become available. While waiting, timeouts (for this thread) are processed.

Parameters:

sem semaphore to wait for

int sys_sem_wait_timeout (sys_sem_t *sem*, u32_t *timeout*)

Wait for a semaphore with timeout (specified in ms)

Parameters:

sem semaphore to wait
timeout timeout in ms (0: wait forever)

Returns:

0 on timeout, 1 otherwise

void sys_timeout (u32_t *msecs*, sys_timeout_handler *h*, void * *arg*)

Create a one-shot timer (aka timeout). Timeouts are processed in the following cases:

- while waiting for a message using **sys_mbox_fetch()**
- while waiting for a semaphore using **sys_sem_wait()** or **sys_sem_wait_timeout()**
- while sleeping using the inbuilt **sys_msleep()**

Parameters:

msecs time in milliseconds after that the timer should expire
h callback function to call when msecs have elapsed
arg argument to pass to the callback function

void sys_untimeout (sys_timeout_handler *h*, void * *arg*)

Go through timeout list (for this task only) and remove the first matching entry, even though the timeout has not triggered yet.

Note:

This function only works as expected if there is only one timeout calling 'h' in the list of timeouts.

Parameters:

h callback function that would be called by the timeout
arg callback argument that would be passed to h

lwip/src/core/tcp.c File Reference

Functions

- void **tcp_tmr** (void)
- err_t **tcp_close** (struct tcp_pcb *pcb)
- void **tcp_abort** (struct tcp_pcb *pcb)
- err_t **tcp_bind** (struct tcp_pcb *pcb, struct ip_addr *ipaddr, u16_t port)
- struct tcp_pcb * **tcp_listen_with_backlog** (struct tcp_pcb *pcb, u8_t backlog)
- void **tcp_recved** (struct tcp_pcb *pcb, u16_t len)
- err_t **tcp_connect** (struct tcp_pcb *pcb, struct ip_addr *ipaddr, u16_t port, err_t(*connected)(void *arg, struct tcp_pcb *tpcb, err_t err))
- void **tcp_slowtmr** (void)
- void **tcp_fasttmr** (void)
- u8_t **tcp_segs_free** (struct tcp_seg *seg)
- u8_t **tcp_seg_free** (struct tcp_seg *seg)
- void **tcp_setprio** (struct tcp_pcb *pcb, u8_t prio)
- struct tcp_seg * **tcp_seg_copy** (struct tcp_seg *seg)
- struct tcp_pcb * **tcp_alloc** (u8_t prio)
- struct tcp_pcb * **tcp_new** (void)
- void **tcp_arg** (struct tcp_pcb *pcb, void *arg)
- void **tcp_recv** (struct tcp_pcb *pcb, err_t(*recv)(void *arg, struct tcp_pcb *tpcb, struct pbuf *p, err_t err))
- void **tcp_sent** (struct tcp_pcb *pcb, err_t(*sent)(void *arg, struct tcp_pcb *tpcb, u16_t len))
- void **tcp_err** (struct tcp_pcb *pcb, void(*errf)(void *arg, err_t err))
- void **tcp_accept** (struct tcp_pcb *pcb, err_t(*accept)(void *arg, struct tcp_pcb *newpcb, err_t err))
- void **tcp_poll** (struct tcp_pcb *pcb, err_t(*poll)(void *arg, struct tcp_pcb *tpcb), u8_t interval)
- void **tcp_pcb_purge** (struct tcp_pcb *pcb)
- void **tcp_pcb_remove** (struct tcp_pcb **pcblist, struct tcp_pcb *pcb)
- u32_t **tcp_next_iss** (void)
- u16_t **tcp_eff_send_mss** (u16_t sendmss, struct ip_addr *addr)
- void **tcp_debug_print** (struct tcp_hdr *tcphdr)
- void **tcp_debug_print_state** (enum tcp_state s)
- void **tcp_debug_print_flags** (u8_t flags)
- void **tcp_debug_print_pcbs** (void)
- s16_t **tcp_pcbs_sane** (void)

Variables

- struct tcp_pcb * **tcp_bound_pcbs**
- union tcp_listen_pcbs_t **tcp_listen_pcbs**
- struct tcp_pcb * **tcp_active_pcbs**
- struct tcp_pcb * **tcp_tw_pcbs**

Detailed Description

Transmission Control Protocol for IP

This file contains common functions for the TCP implementation, such as functinos for manipulating the data structures and the TCP timer functions. TCP functions related to input and output is found in **tcp_in.c** and **tcp_out.c** respectively.

Function Documentation

void tcp_abort (struct tcp_pcb * *pcb*)

Aborts a connection by sending a RST to the remote host and deletes the local protocol control block. This is done when a connection is killed because of shortage of memory.

Parameters:

pcb the tcp_pcb to abort

void tcp_accept (struct tcp_pcb * *pcb*, err_t(*)(void *arg, struct tcp_pcb *newpcb, err_t err) *accept*)

Used for specifying the function that should be called when a LISTENing connection has been connected to another host.

Parameters:

pcb tcp_pcb to set the accept callback
accept callback function to call for this pcb when LISTENing connection has been connected to another host

struct tcp_pcb* tcp_alloc (u8_t *prio*) `[read]`

Allocate a new tcp_pcb structure.

Parameters:

prio priority for the new pcb

Returns:

a new tcp_pcb that initially is in state CLOSED

void tcp_arg (struct tcp_pcb * *pcb*, void * *arg*)

Used to specify the argument that should be passed callback functions.

Parameters:

pcb tcp_pcb to set the callback argument
arg void pointer argument to pass to callback functions

err_t tcp_bind (struct tcp_pcb * *pcb*, struct ip_addr * *ipaddr*, u16_t *port*)

Binds the connection to a local portnumber and IP address. If the IP address is not given (i.e., ipaddr == NULL), the IP address of the outgoing network interface is used instead.

Parameters:

pcb the tcp_pcb to bind (no check is done whether this pcb is already bound!)
ipaddr the local ip address to bind to (use IP_ADDR_ANY to bind to any local address
port the local port to bind to

Returns:

ERR_USE if the port is already in use ERR_OK if bound

err_t tcp_close (struct tcp_pcb * *pcb*)

Closes the connection held by the PCB.

Listening pcbs are freed and may not be referenced any more. Connection pcbs are freed if not yet connected and may not be referenced any more. If a connection is established (at least SYN received or in a closing state), the connection is closed, and put in a closing state. The pcb is then automatically freed in **tcp_slowtmr()**. It is therefore unsafe to reference it.

Parameters:

pcb the tcp_pcb to close

Returns:

ERR_OK if connection has been closed another err_t if closing failed and pcb is not freed

err_t tcp_connect (struct tcp_pcb * *pcb*, struct ip_addr * *ipaddr*, u16_t *port*, err_t(*)(void *arg, struct tcp_pcb *tpcb, err_t err) *connected*)

Connects to another host. The function given as the "connected" argument will be called when the connection has been established.

Parameters:

pcb the tcp_pcb used to establish the connection
ipaddr the remote ip address to connect to
port the remote tcp port to connect to
connected callback function to call when connected (or on error)

Returns:

ERR_VAL if invalid arguments are given ERR_OK if connect request has been sent other err_t values if connect request couldn't be sent

void tcp_debug_print (struct tcp_hdr * *tcphdr*)

Print a tcp header for debugging purposes.

Parameters:

tcphdr pointer to a struct tcp_hdr

void tcp_debug_print_flags (u8_t *flags*)

Print tcp flags for debugging purposes.

Parameters:

flags tcp flags, all active flags are printed

void tcp_debug_print_pcbs (void)

Print all tcp_pcbs in every list for debugging purposes.

void tcp_debug_print_state (enum tcp_state *s*)

Print a tcp state for debugging purposes.

Parameters:

s enum tcp_state to print

u16_t tcp_eff_send_mss (u16_t *sendmss*, struct ip_addr * *addr*)

Calcluates the effective send mss that can be used for a specific IP address by using ip_route to determin the **netif** used to send to the address and calculating the minimum of TCP_MSS and that netif's mtu (if set).

void tcp_err (struct tcp_pcb * *pcb*, void(*)(void *arg, err_t err) *errf*)

Used to specify the function that should be called when a fatal error has occured on the connection.

Parameters:

pcb tcp_pcb to set the err callback
errf callback function to call for this pcb when a fatal error has occured on the connection

void tcp_fasttmr (void)

Is called every TCP_FAST_INTERVAL (250 ms) and process data previously "refused" by upper layer (application) and sends delayed ACKs.

Automatically called from **tcp_tmr()**.

struct tcp_pcb* tcp_listen_with_backlog (struct tcp_pcb * *pcb*, u8_t *backlog*) `[read]`

Set the state of the connection to be LISTEN, which means that it is able to accept incoming connections. The protocol control block is reallocated in order to consume less memory. Setting the connection to LISTEN is an irreversible process.

Parameters:

pcb the original tcp_pcb
backlog the incoming connections queue limit

Returns:

tcp_pcb used for listening, consumes less memory.

Note:

The original tcp_pcb is freed. This function therefore has to be called like this: tpcb = tcp_listen(tpcb);

struct tcp_pcb* tcp_new (void) `[read]`

Creates a new TCP protocol control block but doesn't place it on any of the TCP PCB lists. The pcb is not put on any list until binding using **tcp_bind()**.

u32_t tcp_next_iss (void)

Calculates a new initial sequence number for new connections.

Returns:

u32_t pseudo random sequence number

void tcp_pcb_purge (struct tcp_pcb * *pcb*)

Purges a TCP PCB. Removes any buffered data and frees the buffer memory (pcb->ooseq, pcb->unsent and pcb->unacked are freed).

Parameters:

pcb tcp_pcb to purge. The pcb itself is not deallocated!

void tcp_pcb_remove (struct tcp_pcb ** *pcblist*, struct tcp_pcb * *pcb*)

Purges the PCB and removes it from a PCB list. Any delayed ACKs are sent first.

Parameters:

pcblist PCB list to purge.
pcb tcp_pcb to purge. The pcb itself is also deallocated!

s16_t tcp_pcbs_sane (void)

Check state consistency of the tcp_pcb lists.

void tcp_poll (struct tcp_pcb * *pcb*, err_t(*)(void *arg, struct tcp_pcb *tpcb) *poll*, u8_t *interval*)

Used to specify the function that should be called periodically from TCP. The interval is specified in terms of the TCP coarse timer interval, which is called twice a second.

void tcp_recv (struct tcp_pcb * *pcb*, err_t(*)(void *arg, struct tcp_pcb *tpcb, struct pbuf *p, err_t err) *recv*)

Used to specify the function that should be called when a TCP connection receives data.

Parameters:

pcb tcp_pcb to set the recv callback
recv callback function to call for this pcb when data is received

void tcp_recved (struct tcp_pcb * *pcb*, u16_t *len*)

This function should be called by the application when it has processed the data. The purpose is to advertise a larger window when the data has been processed.

Parameters:

pcb the tcp_pcb for which data is read
len the amount of bytes that have been read by the application

struct tcp_seg* tcp_seg_copy (struct tcp_seg * *seg*) `[read]`

Returns a copy of the given TCP segment. The pbuf and data are not copied, only the pointers

Parameters:

seg the old tcp_seg

Returns:

a copy of seg

u8_t tcp_seg_free (struct tcp_seg * *seg*)

Frees a TCP segment (tcp_seg structure).

Parameters:

seg single tcp_seg to free

Returns:

the number of pbufs that were deallocated

u8_t tcp_segs_free (struct tcp_seg * *seg*)

Deallocates a list of TCP segments (tcp_seg structures).

Parameters:

seg tcp_seg list of TCP segments to free

Returns:

the number of pbufs that were deallocated

void tcp_sent (struct tcp_pcb * *pcb*, err_t(*)(void *arg, struct tcp_pcb *tpcb, u16_t len) *sent*)

Used to specify the function that should be called when TCP data has been successfully delivered to the remote host.

Parameters:

pcb tcp_pcb to set the sent callback
sent callback function to call for this pcb when data is successfully sent

void tcp_setprio (struct tcp_pcb * *pcb*, u8_t *prio*)

Sets the priority of a connection.

Parameters:

pcb the tcp_pcb to manipulate
prio new priority

void tcp_slowtmr (void)

Called every 500 ms and implements the retransmission timer and the timer that removes PCBs that have been in TIME-WAIT for enough time. It also increments various timers such as the inactivity timer in each PCB.

Automatically called from **tcp_tmr**().

void tcp_tmr (void)

Called periodically to dispatch TCP timers.

Variable Documentation

struct tcp_pcb* tcp_active_pcbs

List of all TCP PCBs that are in a state in which they accept or send data.

struct tcp_pcb* tcp_bound_pcbs

List of all TCP PCBs bound but not yet (connected || listening)

union tcp_listen_pcbs_t tcp_listen_pcbs

List of all TCP PCBs in LISTEN state

struct tcp_pcb* tcp_tw_pcbs

List of all TCP PCBs in TIME-WAIT state

lwip/src/core/tcp_in.c File Reference

Functions

- void **tcp_input** (struct pbuf *p, struct **netif** *inp)

Detailed Description

Transmission Control Protocol, incoming traffic

The input processing functions of the TCP layer.

These functions are generally called in the order (**ip_input()** ->) **tcp_input()** -> * tcp_process() -> tcp_receive() (-> application).

Function Documentation

void tcp_input (struct pbuf * *p*, struct netif * *inp*)

The initial input processing of TCP. It verifies the TCP header, demultiplexes the segment between the PCBs and passes it on to tcp_process(), which implements the TCP finite state machine. This function is called by the IP layer (in **ip_input()**).

Parameters:

p received TCP segment to process (p->payload pointing to the IP header)
inp network interface on which this segment was received

lwip/src/core/tcp_out.c File Reference

Functions

- err_t **tcp_send_ctrl** (struct tcp_pcb *pcb, u8_t flags)
- err_t **tcp_write** (struct tcp_pcb *pcb, const void *data, u16_t len, u8_t apiflags)
- err_t **tcp_enqueue** (struct tcp_pcb *pcb, void *arg, u16_t len, u8_t flags, u8_t apiflags, u8_t *optdata, u8_t optlen)
- err_t **tcp_output** (struct tcp_pcb *pcb)
- void **tcp_rst** (u32_t seqno, u32_t ackno, struct ip_addr *local_ip, struct ip_addr *remote_ip, u16_t local_port, u16_t remote_port)
- void **tcp_rexmit_rto** (struct tcp_pcb *pcb)
- void **tcp_rexmit** (struct tcp_pcb *pcb)
- void **tcp_keepalive** (struct tcp_pcb *pcb)
- void **tcp_zero_window_probe** (struct tcp_pcb *pcb)

Detailed Description

Transmission Control Protocol, outgoing traffic

The output functions of TCP.

Function Documentation

err_t tcp_enqueue (struct tcp_pcb * *pcb*, void * *arg*, u16_t *len*, u8_t *flags*, u8_t *apiflags*, u8_t * *optdata*, u8_t *optlen*)

Enqueue either data or TCP options (but not both) for tranmission

Called by **tcp_connect**(), tcp_listen_input(), **tcp_send_ctrl**() and **tcp_write**().

Parameters:

pcb Protocol control block for the TCP connection to enqueue data for.
arg Pointer to the data to be enqueued for sending.
len Data length in bytes
flags tcp header flags to set in the outgoing segment
apiflags combination of following flags :

- TCP_WRITE_FLAG_COPY (0x01) data will be copied into memory belonging to the stack
- TCP_WRITE_FLAG_MORE (0x02) for TCP connection, PSH flag will be set on last segment sent,

optdata
optlen

void tcp_keepalive (struct tcp_pcb * *pcb*)

Send keepalive packets to keep a connection active although no data is sent over it.

Called by **tcp_slowtmr**()

Parameters:

pcb the tcp_pcb for which to send a keepalive packet

err_t tcp_output (struct tcp_pcb * *pcb*)

Find out what we can send and send it

Parameters:

pcb Protocol control block for the TCP connection to send data

Returns:

ERR_OK if data has been sent or nothing to send another err_t on error

void tcp_rexmit (struct tcp_pcb * *pcb*)

Requeue the first unacked segment for retransmission

Called by tcp_receive() for fast retramsmit.

Parameters:

pcb the tcp_pcb for which to retransmit the first unacked segment

void tcp_rexmit_rto (struct tcp_pcb * *pcb*)

Requeue all unacked segments for retransmission

Called by **tcp_slowtmr()** for slow retransmission.

Parameters:

pcb the tcp_pcb for which to re-enqueue all unacked segments

void tcp_rst (u32_t *seqno*, u32_t *ackno*, struct ip_addr * *local_ip*, struct ip_addr * *remote_ip*, u16_t *local_port*, u16_t *remote_port*)

Send a TCP RESET packet (empty segment with RST flag set) either to abort a connection or to show that there is no matching local connection for a received segment.

Called by **tcp_abort()** (to abort a local connection), **tcp_input()** (if no matching local pcb was found), tcp_listen_input() (if incoming segment has ACK flag set) and tcp_process() (received segment in the wrong state)

Since a RST segment is in most cases not sent for an active connection, **tcp_rst()** has a number of arguments that are taken from a tcp_pcb for most other segment output functions.

Parameters:

seqno the sequence number to use for the outgoing segment
ackno the acknowledge number to use for the outgoing segment
local_ip the local IP address to send the segment from
remote_ip the remote IP address to send the segment to
local_port the local TCP port to send the segment from
remote_port the remote TCP port to send the segment to

err_t tcp_send_ctrl (struct tcp_pcb * *pcb*, u8_t *flags*)

Called by **tcp_close()** to send a segment including flags but not data.

Parameters:

pcb the tcp_pcb over which to send a segment
flags the flags to set in the segment header

Returns:

ERR_OK if sent, another err_t otherwise

err_t tcp_write (struct tcp_pcb * *pcb*, const void * *data*, u16_t *len*, u8_t *apiflags*)

Write data for sending (but does not send it immediately).

It waits in the expectation of more data being sent soon (as it can send them more efficiently by combining them together). To prompt the system to send data now, call **tcp_output()** after calling **tcp_write()**.

Parameters:

pcb Protocol control block of the TCP connection to enqueue data for.
data pointer to the data to send
len length (in bytes) of the data to send
apiflags combination of following flags :

- TCP_WRITE_FLAG_COPY (0x01) data will be copied into memory belonging to the stack
- TCP_WRITE_FLAG_MORE (0x02) for TCP connection, PSH flag will be set on last segment sent,

Returns:

ERR_OK if enqueued, another err_t on error

See also:

tcp_write()

void tcp_zero_window_probe (struct tcp_pcb * *pcb*)

Send persist timer zero-window probes to keep a connection active when a window update is lost.

Called by **tcp_slowtmr()**

Parameters:

pcb the tcp_pcb for which to send a zero-window probe packet

lwip/src/core/udp.c File Reference

Functions

- void **udp_input** (struct pbuf *p, struct **netif** *inp)
- err_t **udp_send** (struct udp_pcb *pcb, struct pbuf *p)
- err_t **udp_sendto** (struct udp_pcb *pcb, struct pbuf *p, struct ip_addr *dst_ip, u16_t dst_port)
- err_t **udp_sendto_if** (struct udp_pcb *pcb, struct pbuf *p, struct ip_addr *dst_ip, u16_t dst_port, struct **netif** ***netif**)
- err_t **udp_bind** (struct udp_pcb *pcb, struct ip_addr *ipaddr, u16_t port)
- err_t **udp_connect** (struct udp_pcb *pcb, struct ip_addr *ipaddr, u16_t port)
- void **udp_disconnect** (struct udp_pcb *pcb)
- void **udp_recv** (struct udp_pcb *pcb, void(*recv)(void *arg, struct udp_pcb *upcb, struct pbuf *p, struct ip_addr *addr, u16_t port), void *recv_arg)
- void **udp_remove** (struct udp_pcb *pcb)
- struct udp_pcb * **udp_new** (void)
- void **udp_debug_print** (struct udp_hdr *udphdr)

Detailed Description

User Datagram Protocol module

Function Documentation

err_t udp_bind (struct udp_pcb * *pcb*, struct ip_addr * *ipaddr*, u16_t *port*)

Bind an UDP PCB.

Parameters:

pcb UDP PCB to be bound with a local address ipaddr and port.
ipaddr local IP address to bind with. Use IP_ADDR_ANY to bind to all local interfaces.
port local UDP port to bind with. Use 0 to automatically bind to a random port between UDP_LOCAL_PORT_RANGE_START and UDP_LOCAL_PORT_RANGE_END.

ipaddr & port are expected to be in the same byte order as in the pcb.

Returns:

lwIP error code.

- ERR_OK. Successful. No error occured.
- ERR_USE. The specified ipaddr and port are already bound to by another UDP PCB.

See also:

udp_disconnect()

err_t udp_connect (struct udp_pcb * *pcb*, struct ip_addr * *ipaddr*, u16_t *port*)

Connect an UDP PCB.

This will associate the UDP PCB with the remote address.

Parameters:

pcb UDP PCB to be connected with remote address ipaddr and port.
ipaddr remote IP address to connect with.
port remote UDP port to connect with.

Returns:

lwIP error code

ipaddr & port are expected to be in the same byte order as in the pcb.

The udp pcb is bound to a random local port if not already bound.

See also:

udp_disconnect()

TODO: this functionality belongs in upper layers

TODO: this will bind the udp pcb locally, to the interface which is used to route output packets to the remote address. However, we might want to accept incoming packets on any interface!

void udp_debug_print (struct udp_hdr * *udphdr*)

Print UDP header information for debug purposes.

Parameters:

udphdr pointer to the udp header in memory.

void udp_disconnect (struct udp_pcb * *pcb*)

Disconnect a UDP PCB

Parameters:

pcb the udp pcb to disconnect.

void udp_input (struct pbuf * *p*, struct netif * *inp*)

Process an incoming UDP datagram.

Given an incoming UDP datagram (as a chain of pbufs) this function finds a corresponding UDP PCB and hands over the pbuf to the pcbs recv function. If no pcb is found or the datagram is incorrect, the pbuf is freed.

Parameters:

p pbuf to be demultiplexed to a UDP PCB.
inp network interface on which the datagram was received.

struct udp_pcb* udp_new (void) `[read]`

Create a UDP PCB.

Returns:

The UDP PCB which was created. NULL if the PCB data structure could not be allocated.

See also:

udp_remove()

void udp_recv (struct udp_pcb * *pcb*, void(*)(void *arg, struct udp_pcb *upcb, struct pbuf *p, struct ip_addr *addr, u16_t port) *recv*, void * *recv_arg*)

Set a receive callback for a UDP PCB

This callback will be called when receiving a datagram for the pcb.

Parameters:

pcb the pcb for wich to set the recv callback
recv function pointer of the callback function
recv_arg additional argument to pass to the callback function

void udp_remove (struct udp_pcb * *pcb*)

Remove an UDP PCB.

Parameters:

pcb UDP PCB to be removed. The PCB is removed from the list of UDP PCB's and the data structure is freed from memory.

See also:

udp_new()

err_t udp_send (struct udp_pcb * *pcb*, struct pbuf * *p*)

Send data using UDP.

Parameters:

pcb UDP PCB used to send the data.
p chain of pbuf's to be sent.

The datagram will be sent to the current remote_ip & remote_port stored in pcb. If the pcb is not bound to a port, it will automatically be bound to a random port.

Returns:

lwIP error code.

- ERR_OK. Successful. No error occured.
- ERR_MEM. Out of memory.
- ERR_RTE. Could not find route to destination address.
- More errors could be returned by lower protocol layers.

See also:

udp_disconnect() udp_sendto()

err_t udp_sendto (struct udp_pcb * *pcb*, struct pbuf * *p*, struct ip_addr * *dst_ip*, u16_t *dst_port*)

Send data to a specified address using UDP.

Parameters:

pcb UDP PCB used to send the data.
p chain of pbuf's to be sent.
dst_ip Destination IP address.
dst_port Destination UDP port.

dst_ip & dst_port are expected to be in the same byte order as in the pcb.

If the PCB already has a remote address association, it will be restored after the data is sent.

Returns:

lwIP error code (

See also:

udp_send for possible error codes)
udp_disconnect() udp_send()

err_t udp_sendto_if (struct udp_pcb * *pcb*, struct pbuf * *p*, struct ip_addr * *dst_ip*, u16_t *dst_port*, struct netif * *netif*)

Send data to a specified address using UDP. The **netif** used for sending can be specified.

This function exists mainly for DHCP, to be able to send UDP packets on a **netif** that is still down.

Parameters:

pcb UDP PCB used to send the data.
p chain of pbuf's to be sent.

dst_ip Destination IP address.
dst_port Destination UDP port.
netif the **netif** used for sending.

dst_ip & dst_port are expected to be in the same byte order as in the pcb.

Returns:

lwIP error code (

See also:

udp_send for possible error codes)
udp_disconnect() udp_send()

lwip/src/include/ipv4/lwip/autoip.h File Reference

Functions

- void **autoip_init** (void)
- err_t **autoip_start** (struct **netif** ***netif**)
- err_t **autoip_stop** (struct **netif** ***netif**)
- void **autoip_arp_reply** (struct **netif** ***netif**, struct **etharp_hdr** *hdr)
- void **autoip_tmr** (void)

Detailed Description

AutoIP Automatic LinkLocal IP Configuration

Function Documentation

void autoip_arp_reply (struct netif * *netif*, struct etharp_hdr * *hdr*)

Handles every incoming ARP Packet, called by etharp_arp_input

Handles every incoming ARP Packet, called by etharp_arp_input.

Parameters:

netif network interface to use for autoip processing
hdr Incoming ARP packet

void autoip_init (void)

Init srand, has to be called before entering mainloop

Initialize this module

err_t autoip_start (struct netif * *netif*)

Start AutoIP client

Start AutoIP client

Parameters:

netif network interface on which start the AutoIP client

err_t autoip_stop (struct netif * *netif*)

Stop AutoIP client

Stop AutoIP client

Parameters:

netif network interface on which stop the AutoIP client

void autoip_tmr (void)

Has to be called in loop every AUTOIP_TMR_INTERVAL milliseconds

Has to be called in loop every AUTOIP_TMR_INTERVAL milliseconds

lwip/src/include/lwip/dhcp.h File Reference

Data Structures

- struct **dhcp_msg**

Functions

- PACK_STRUCT_END err_t **dhcp_start** (struct **netif** *__netif__)
- err_t **dhcp_renew** (struct **netif** *__netif__)
- err_t **dhcp_release** (struct **netif** *__netif__)
- void **dhcp_stop** (struct **netif** *__netif__)
- void **dhcp_inform** (struct **netif** *__netif__)
- void **dhcp_arp_reply** (struct **netif** *__netif__, struct ip_addr *addr)
- void **dhcp_coarse_tmr** (void)
- void **dhcp_fine_tmr** (void)

Detailed Description

Function Documentation

void dhcp_arp_reply (struct netif * *netif*, struct ip_addr * *addr*)

if enabled, check whether the offered IP address is not in use, using ARP

Match an ARP reply with the offered IP address.

Parameters:

netif the network interface on which the reply was received
addr The IP address we received a reply from

void dhcp_coarse_tmr (void)

to be called every minute

The DHCP timer that checks for lease renewal/rebind timeouts.

void dhcp_fine_tmr (void)

to be called every half second

DHCP transaction timeout handling

A DHCP server is expected to respond within a short period of time. This timer checks whether an outstanding DHCP request is timed out.

void dhcp_inform (struct netif * *netif*)

inform server of our manual IP address

Inform a DHCP server of our manual configuration.

This informs DHCP servers of our fixed IP address configuration by sending an INFORM message. It does not involve DHCP address configuration, it is just here to be nice to the network.

Parameters:

netif The lwIP network interface

err_t dhcp_release (struct netif * *netif*)

release the DHCP lease, usually called before **dhcp_stop()**

Release a DHCP lease.

Parameters:

netif network interface which must release its lease

err_t dhcp_renew (struct netif * *netif*)

enforce early lease renewal (not needed normally)

Renew an existing DHCP lease at the involved DHCP server.

Parameters:

netif network interface which must renew its lease

PACK_STRUCT_END err_t dhcp_start (struct netif * *netif*)

start DHCP configuration

Start DHCP negotiation for a network interface.

If no DHCP client instance was attached to this interface, a new client is created first. If a DHCP client instance was already present, it restarts negotiation.

Parameters:

netif The lwIP network interface

Returns:

lwIP error code

- ERR_OK - No error
- ERR_MEM - Out of memory

void dhcp_stop (struct netif * *netif*)

stop DHCP configuration

Remove the DHCP client from the interface.

Parameters:

netif The network interface to stop DHCP on

lwip/src/include/lwip/opt.h File Reference

Detailed Description

lwIP Options Configuration

lwip/src/include/lwip/snmp_asn1.h File Reference

Functions

- err_t **snmp_asn1_dec_type** (struct pbuf *p, u16_t ofs, u8_t *type)
- err_t **snmp_asn1_dec_length** (struct pbuf *p, u16_t ofs, u8_t *octets_used, u16_t *length)
- err_t **snmp_asn1_dec_u32t** (struct pbuf *p, u16_t ofs, u16_t len, u32_t *value)
- err_t **snmp_asn1_dec_s32t** (struct pbuf *p, u16_t ofs, u16_t len, s32_t *value)
- err_t **snmp_asn1_dec_oid** (struct pbuf *p, u16_t ofs, u16_t len, struct **snmp_obj_id** *oid)
- err_t **snmp_asn1_dec_raw** (struct pbuf *p, u16_t ofs, u16_t len, u16_t raw_len, u8_t *raw)
- void **snmp_asn1_enc_length_cnt** (u16_t length, u8_t *octets_needed)
- void **snmp_asn1_enc_u32t_cnt** (u32_t value, u16_t *octets_needed)
- void **snmp_asn1_enc_s32t_cnt** (s32_t value, u16_t *octets_needed)
- void **snmp_asn1_enc_oid_cnt** (u8_t ident_len, s32_t *ident, u16_t *octets_needed)
- err_t **snmp_asn1_enc_type** (struct pbuf *p, u16_t ofs, u8_t type)
- err_t **snmp_asn1_enc_length** (struct pbuf *p, u16_t ofs, u16_t length)
- err_t **snmp_asn1_enc_u32t** (struct pbuf *p, u16_t ofs, u8_t octets_needed, u32_t value)
- err_t **snmp_asn1_enc_s32t** (struct pbuf *p, u16_t ofs, u8_t octets_needed, s32_t value)
- err_t **snmp_asn1_enc_oid** (struct pbuf *p, u16_t ofs, u8_t ident_len, s32_t *ident)
- err_t **snmp_asn1_enc_raw** (struct pbuf *p, u16_t ofs, u8_t raw_len, u8_t *raw)

Detailed Description

Abstract Syntax Notation One (ISO 8824, 8825) codec.

Function Documentation

err_t snmp_asn1_dec_length (struct pbuf * *p*, u16_t *ofs*, u8_t * *octets_used*, u16_t * *length*)

Decodes length field from incoming pbuf chain into host length.

Parameters:

p points to a pbuf holding an ASN1 coded length
ofs points to the offset within the pbuf chain of the ASN1 coded length
octets_used returns number of octets used by the length code
length return host order length, upto 64k

Returns:

ERR_OK if successfull, ERR_ARG if we can't (or won't) decode

Todo:

: do we need to accept inefficient codings with many leading zero's?

err_t snmp_asn1_dec_oid (struct pbuf * *p*, u16_t *ofs*, u16_t *len*, struct snmp_obj_id * *oid*)

Decodes object identifier from incoming message into array of s32_t.

Parameters:

p points to a pbuf holding an ASN1 coded object identifier
ofs points to the offset within the pbuf chain of the ASN1 coded object identifier
len length of the coded object identifier
oid return object identifier struct

Returns:

ERR_OK if successfull, ERR_ARG if we can't (or won't) decode

err_t snmp_asn1_dec_raw (struct pbuf * *p*, u16_t *ofs*, u16_t *len*, u16_t *raw_len*, u8_t * *raw*)

Decodes (copies) raw data (ip-addresses, octet strings, opaque encoding) from incoming message into array.

Parameters:

p points to a pbuf holding an ASN1 coded raw data
ofs points to the offset within the pbuf chain of the ASN1 coded raw data
len length of the coded raw data (zero is valid, e.g. empty string!)
raw_len length of the raw return value
raw return raw bytes

Returns:

ERR_OK if successfull, ERR_ARG if we can't (or won't) decode

err_t snmp_asn1_dec_s32t (struct pbuf * *p*, u16_t *ofs*, u16_t *len*, s32_t * *value*)

Decodes integer into s32_t.

Parameters:

p points to a pbuf holding an ASN1 coded integer
ofs points to the offset within the pbuf chain of the ASN1 coded integer
len length of the coded integer field
value return host order integer

Returns:

ERR_OK if successfull, ERR_ARG if we can't (or won't) decode

Note:

ASN coded integers are _always_ signed!

err_t snmp_asn1_dec_type (struct pbuf * *p*, u16_t *ofs*, u8_t * *type*)

Retrieves type field from incoming pbuf chain.

Parameters:

p points to a pbuf holding an ASN1 coded type field
ofs points to the offset within the pbuf chain of the ASN1 coded type field
type return ASN1 type

Returns:

ERR_OK if successfull, ERR_ARG if we can't (or won't) decode

err_t snmp_asn1_dec_u32t (struct pbuf * *p*, u16_t *ofs*, u16_t *len*, u32_t * *value*)

Decodes positive integer (counter, gauge, timeticks) into u32_t.

Parameters:

p points to a pbuf holding an ASN1 coded integer
ofs points to the offset within the pbuf chain of the ASN1 coded integer
len length of the coded integer field
value return host order integer

Returns:

ERR_OK if successfull, ERR_ARG if we can't (or won't) decode

Note:

ASN coded integers are _always_ signed. E.g. +0xFFFF is coded as 0x00,0xFF,0xFF. Note the leading sign octet. A positive value of 0xFFFFFFFF is preceded with 0x00 and the length is 5 octets!!

err_t snmp_asn1_enc_length (struct pbuf * *p*, u16_t *ofs*, u16_t *length*)

Encodes host order length field into a pbuf chained ASN1 msg.

Parameters:

p points to output pbuf to encode length into
ofs points to the offset within the pbuf chain
length is the host order length to be encoded

Returns:

ERR_OK if successfull, ERR_ARG if we can't (or won't) encode

void snmp_asn1_enc_length_cnt (u16_t *length*, u8_t * *octets_needed*)

Returns octet count for length.

Parameters:

length
octets_needed points to the return value

err_t snmp_asn1_enc_oid (struct pbuf * *p*, u16_t *ofs*, u8_t *ident_len*, s32_t * *ident*)

Encodes object identifier into a pbuf chained ASN1 msg.

Parameters:

p points to output pbuf to encode oid into
ofs points to the offset within the pbuf chain
ident_len object identifier array length
ident points to object identifier array

Returns:

ERR_OK if successfull, ERR_ARG if we can't (or won't) encode

void snmp_asn1_enc_oid_cnt (u8_t *ident_len*, s32_t * *ident*, u16_t * *octets_needed*)

Returns octet count for an object identifier.

Parameters:

ident_len object identifier array length
ident points to object identifier array
octets_needed points to the return value

err_t snmp_asn1_enc_raw (struct pbuf * *p*, u16_t *ofs*, u8_t *raw_len*, u8_t * *raw*)

Encodes raw data (octet string, opaque) into a pbuf chained ASN1 msg.

Parameters:

p points to output pbuf to encode raw data into
ofs points to the offset within the pbuf chain
raw_len raw data length
raw points raw data

Returns:

ERR_OK if successfull, ERR_ARG if we can't (or won't) encode

err_t snmp_asn1_enc_s32t (struct pbuf * *p*, u16_t *ofs*, u8_t *octets_needed*, s32_t *value*)

Encodes s32_t integer into a pbuf chained ASN1 msg.

Parameters:

p points to output pbuf to encode value into
ofs points to the offset within the pbuf chain
octets_needed encoding length (from **snmp_asn1_enc_s32t_cnt()**)
value is the host order s32_t value to be encoded

Returns:

ERR_OK if successfull, ERR_ARG if we can't (or won't) encode

See also:

snmp_asn1_enc_s32t_cnt()

void snmp_asn1_enc_s32t_cnt (s32_t *value*, u16_t * *octets_needed*)

Returns octet count for an s32_t.

Parameters:

value
octets_needed points to the return value

Note:

ASN coded integers are _always_ signed.

err_t snmp_asn1_enc_type (struct pbuf * *p*, u16_t *ofs*, u8_t *type*)

Encodes ASN type field into a pbuf chained ASN1 msg.

Parameters:

p points to output pbuf to encode value into
ofs points to the offset within the pbuf chain
type input ASN1 type

Returns:

ERR_OK if successfull, ERR_ARG if we can't (or won't) encode

err_t snmp_asn1_enc_u32t (struct pbuf * *p*, u16_t *ofs*, u8_t *octets_needed*, u32_t *value*)

Encodes u32_t (counter, gauge, timeticks) into a pbuf chained ASN1 msg.

Parameters:

p points to output pbuf to encode value into
ofs points to the offset within the pbuf chain
octets_needed encoding length (from **snmp_asn1_enc_u32t_cnt()**)
value is the host order u32_t value to be encoded

Returns:

ERR_OK if successfull, ERR_ARG if we can't (or won't) encode

See also:

snmp_asn1_enc_u32t_cnt()

void snmp_asn1_enc_u32t_cnt (u32_t *value*, u16_t * *octets_needed*)

Returns octet count for an u32_t.

Parameters:

value
octets_needed points to the return value

Note:

ASN coded integers are _always_ signed. E.g. +0xFFFF is coded as 0x00,0xFF,0xFF. Note the leading sign octet. A positive value of 0xFFFFFFFF is preceded with 0x00 and the length is 5 octets!!

lwip/src/include/lwip/snmp_msg.h File Reference

Data Structures

- struct **snmp_resp_header_lengths**
- struct **snmp_trap_header_lengths**

Functions

- void **snmp_init** (void)
- void **snmp_trap_dst_enable** (u8_t dst_idx, u8_t enable)
- void **snmp_trap_dst_ip_set** (u8_t dst_idx, struct ip_addr *dst)
- struct snmp_varbind * **snmp_varbind_alloc** (struct **snmp_obj_id** *oid, u8_t type, u8_t len)
- void **snmp_msg_event** (u8_t request_id)
- err_t **snmp_send_response** (struct snmp_msg_pstat *m_stat)
- err_t **snmp_send_trap** (s8_t generic_trap, struct **snmp_obj_id** *eoid, s32_t specific_trap)

Variables

- const s32_t **snmp_version**
- const char **snmp_publiccommunity** [7]
- struct snmp_msg_trap **trap_msg**

Detailed Description

SNMP Agent message handling structures.

Function Documentation

void snmp_init (void)

Agent setup, start listening to port 161.

Starts SNMP Agent. Allocates UDP pcb and binds it to IP_ADDR_ANY port 161.

void snmp_msg_event (u8_t *request_id*)

Handle an internal (recv) or external (private response) event.

Handle one internal or external event. Called for one async event. (recv external/private answer)

Parameters:

request_id identifies requests from 0 to (SNMP_CONCURRENT_REQUESTS-1)

err_t snmp_send_response (struct snmp_msg_pstat * *m_stat*)

Sends a 'getresponse' message to the request originator.

Parameters:

m_stat points to the current message request state source

Returns:

ERR_OK when success, ERR_MEM if we're out of memory

Note:

the caller is responsible for filling in outvb in the m_stat and provide error-status and index (except for tooBig errors) ...

Todo:

do we need separate rx and tx pcbs for threaded case?

connect to the originating source

Todo:

release some memory, retry and return tooBig? tooMuchHassle?

disassociate remote address and port with this pcb

err_t snmp_send_trap (s8_t *generic_trap*, struct snmp_obj_id * *eoid*, s32_t *specific_trap*)

Sends an generic or enterprise specific trap message.

Parameters:

generic_trap is the trap code
eoid points to enterprise object identifier
specific_trap used for enterprise traps when generic_trap == 6

Returns:

ERR_OK when success, ERR_MEM if we're out of memory

Note:

the caller is responsible for filling in outvb in the trap_msg
the use of the enterpise identifier field is per RFC1215. Use .iso.org.dod.internet.mgmt.mib-2.snmp for generic traps and .iso.org.dod.internet.private.enterprises.yourenterprise (sysObjectID) for specific traps.

connect to the TRAP destination

disassociate remote address and port with this pcb

void snmp_trap_dst_enable (u8_t *dst_idx*, u8_t *enable*)

Sets enable switch for this trap destination.

Parameters:

dst_idx index in 0 .. SNMP_TRAP_DESTINATIONS-1
enable switch if 0 destination is disabled >0 enabled.

void snmp_trap_dst_ip_set (u8_t *dst_idx*, struct ip_addr * *dst*)

Sets IPv4 address for this trap destination.

Parameters:

dst_idx index in 0 .. SNMP_TRAP_DESTINATIONS-1
dst IPv4 address in host order.

struct snmp_varbind* snmp_varbind_alloc (struct snmp_obj_id * *oid*, u8_t *type*, u8_t *len*) `[read]`

Varbind-list functions.

Variable Documentation

const char snmp_publiccommunity[7]

Agent default "public" community string

default SNMP community string

const s32_t snmp_version

Agent Version constant, 0 = v1 oddity

SNMP v1 == 0

struct snmp_msg_trap trap_msg

TRAP message structure

lwip/src/include/lwip/snmp_structs.h File Reference

Data Structures

- struct **obj_def**
- struct **mib_node**
- struct **mib_array_node**
- struct **mib_ram_array_node**
- struct **mib_list_rootnode**
- struct **mib_external_node**

Typedefs

- typedef struct **mib_node mib_scalar_node**

Functions

- void **noleafs_get_object_def** (u8_t ident_len, s32_t *ident, struct **obj_def** *od)
- void **snmp_oidtoip** (s32_t *ident, struct ip_addr *ip)
- void **snmp_iptooid** (struct ip_addr *ip, s32_t *ident)
- void **snmp_ifindextonetif** (s32_t ifindex, struct **netif** ****netif**)
- void **snmp_netiftoifindex** (struct **netif** ***netif**, s32_t *ifidx)
- s8_t **snmp_mib_node_insert** (struct **mib_list_rootnode** *rn, s32_t objid, struct mib_list_node **insn)
- s8_t **snmp_mib_node_find** (struct **mib_list_rootnode** *rn, s32_t objid, struct mib_list_node **fn)
- struct **mib_list_rootnode** * **snmp_mib_node_delete** (struct **mib_list_rootnode** *rn, struct mib_list_node *n)
- struct **mib_node** * **snmp_search_tree** (struct **mib_node** *node, u8_t ident_len, s32_t *ident, struct snmp_name_ptr *np)
- struct **mib_node** * **snmp_expand_tree** (struct **mib_node** *node, u8_t ident_len, s32_t *ident, struct **snmp_obj_id** *oidret)
- u8_t **snmp_iso_prefix_tst** (u8_t ident_len, s32_t *ident)
- u8_t **snmp_iso_prefix_expand** (u8_t ident_len, s32_t *ident, struct **snmp_obj_id** *oidret)

Variables

- struct **mib_array_node internet**

Detailed Description

Generic MIB tree structures.

Todo:

namespace prefixes

Typedef Documentation

typedef struct mib_node mib_scalar_node

derived node for scalars .0 index

Function Documentation

void noleafs_get_object_def (u8_t *ident_len*, s32_t * *ident*, struct obj_def * *od*)

dummy function pointers for non-leaf MIB nodes from **mib2.c**

struct mib_node* snmp_expand_tree (struct mib_node * *node*, u8_t *ident_len*, s32_t * *ident*, struct snmp_obj_id * *oidret*) `[read]`

Tree expansion.

void snmp_ifindextonetif (s32_t *ifindex*, struct netif ** *netif*)

Conversion from ifIndex to lwIP **netif**

Parameters:

ifindex is a s32_t object sub-identifier
netif points to returned **netif** struct pointer

void snmp_iptooid (struct ip_addr * *ip*, s32_t * *ident*)

Conversion from lwIP ip_addr to oid

Parameters:

ip points to input struct
ident points to s32_t ident[4] output

u8_t snmp_iso_prefix_expand (u8_t *ident_len*, s32_t * *ident*, struct snmp_obj_id * *oidret*)

Expands object identifier to the **iso.org.dod.internet** prefix for use in getnext operation.

Parameters:

ident_len the length of the supplied object identifier
ident points to the array of sub identifiers
oidret points to returned expanded object identifier

Returns:

1 if it matches, 0 otherwise

Note:

ident_len 0 is allowed, expanding to the first known object id!!

u8_t snmp_iso_prefix_tst (u8_t *ident_len*, s32_t * *ident*)

Test object identifier for the **iso.org.dod.internet** prefix.

Parameters:

ident_len the length of the supplied object identifier
ident points to the array of sub identifiers

Returns:

1 if it matches, 0 otherwise

struct mib_list_rootnode* snmp_mib_node_delete (struct mib_list_rootnode * *rn*, struct mib_list_node * *n*) `[read]`

Removes node from idx list if it has a single child left.

Parameters:

rn points to the root node
n points to the node to delete

Returns:

the nptr to be freed by caller

s8_t snmp_mib_node_find (struct mib_list_rootnode * *rn*, s32_t *objid*, struct mib_list_node ** *fn*)

Finds node in idx list and returns deletion mark.

Parameters:

rn points to the root node
objid is the object sub identifier
fn returns pointer to found node

Returns:

0 if not found, 1 if deletable, 2 can't delete (2 or more children), 3 not a list_node

s8_t snmp_mib_node_insert (struct mib_list_rootnode * *rn*, s32_t *objid*, struct mib_list_node ** *insn*)

Inserts node in idx list in a sorted (ascending order) fashion and allocates the node if needed.

Parameters:

rn points to the root node
objid is the object sub identifier
insn points to a pointer to the inserted node used for constructing the tree.

Returns:

-1 if failed, 1 if inserted, 2 if present.

void snmp_netiftoifindex (struct netif * *netif*, s32_t * *ifidx*)

Conversion from lwIP **netif** to ifIndex

Parameters:

netif points to a **netif** struct
ifidx points to s32_t object sub-identifier

void snmp_oidtoip (s32_t * *ident*, struct ip_addr * *ip*)

Conversion from oid to lwIP ip_addr

Parameters:

ident points to s32_t ident[4] input
ip points to output struct

struct mib_node* snmp_search_tree (struct mib_node * *node*, u8_t *ident_len*, s32_t * *ident*, struct snmp_name_ptr * *np*) `[read]`

Searches tree for the supplied (scalar?) object identifier.

Parameters:

node points to the root of the tree ('.internet')
ident_len the length of the supplied object identifier
ident points to the array of sub identifiers
np points to the found object instance (rerurn)

Returns:

pointer to the requested parent (!) node if success, NULL otherwise

---

Variable Documentation

struct mib_array_node internet `[read]`

export MIB tree from **mib2.c**

lwip/src/netif/etharp.c File Reference

Functions

- void **etharp_tmr** (void)

Detailed Description

Address Resolution Protocol module for IP over Ethernet

Functionally, ARP is divided into two parts. The first maps an IP address to a physical address when sending a packet, and the second part answers requests from other machines for our physical address.

This implementation complies with RFC 826 (Ethernet ARP). It supports Gratuitious ARP from RFC3220 (IP Mobility Support for IPv4) section 4.6 if an interface calls etharp_query(our_netif, its_ip_addr, NULL) upon address change.

Function Documentation

void etharp_tmr (void)

Clears expired entries in the ARP table.

This function should be called every ETHARP_TMR_INTERVAL microseconds (5 seconds), in order to expire entries in the ARP table.

lwip/src/netif/ethernetif.c File Reference

Data Structures

- struct **ethernetif**

Functions

- err_t **ethernetif_init** (struct **netif** ***netif**)

Detailed Description

Ethernet Interface Skeleton

Function Documentation

err_t ethernetif_init (struct netif * *netif*)

Should be called at the beginning of the program to set up the network interface. It calls the function low_level_init() to do the actual setup of the hardware.

This function should be passed as a parameter to **netif_add**().

Parameters:

netif the lwip network interface structure for this **ethernetif**

Returns:

ERR_OK if the loopif is initialized ERR_MEM if private data couldn't be allocated any other err_t on error

lwip/src/netif/loopif.c File Reference

Functions

- void **loopif_poll** (struct **netif** ***netif**)
- err_t **loopif_init** (struct **netif** ***netif**)

Detailed Description

Loop Interface

Function Documentation

err_t loopif_init (struct netif * *netif*)

Initialize a lwip network interface structure for a loopback interface

Parameters:

netif the lwip network interface structure for this loopif

Returns:

ERR_OK if the loopif is initialized ERR_MEM if private data couldn't be allocated

void loopif_poll (struct netif * *netif*)

Call **loopif_poll()** in the main loop of your application. This is to prevent reentering non-reentrant functions like **tcp_input()**. Packets passed to loopif_output() are put on a list that is passed to netif->input() by **loopif_poll()**.

Parameters:

netif the lwip network interface structure for this loopif

lwip/src/netif/slipif.c File Reference

Functions

- err_t **slipif_output** (struct **netif** ***netif**, struct pbuf *p, struct ip_addr *ipaddr)
- err_t **slipif_init** (struct **netif** ***netif**)

Detailed Description

SLIP Interface

Function Documentation

err_t slipif_init (struct netif * *netif*)

SLIP **netif** initialization

Call the arch specific sio_open and remember the opened device in the state field of the **netif**.

Parameters:

netif the lwip network interface structure for this slipif

Returns:

ERR_OK if serial line could be opened, ERR_IF is serial line couldn't be opened

Note:

netif->num must contain the number of the serial port to open (0 by default)

err_t slipif_output (struct netif * *netif*, struct pbuf * *p*, struct ip_addr * *ipaddr*)

Send a pbuf doing the necessary SLIP encapsulation

Uses the serial layer's sio_send()

Parameters:

netif the lwip network interface structure for this slipif
p the pbuf chaing packet to send
ipaddr the ip address to send the packet to (not used for slipif)

Returns:

always returns ERR_OK since the serial layer does not provide return values

lwIP 1.3.0 Page Documentation

Todo List

Global igmp_ip_output_if
should be shared with **ip.c** - ip_output_if

Global igmp_joingroup
undo any other **netif** already joined

Global igmp_start_timer
Important !! this should be random 0 -> max_time. Find out how to do this

Global mem_malloc
: we could try a bigger pool if this one is empty!

File asn1_dec.c
not optimised (yet), favor correctness over speed, favor speed over size

Global snmp_asn1_dec_length
: do we need to accept inefficient codings with many leading zero's?

File asn1_enc.c
not optimised (yet), favor correctness over speed, favor speed over size

Global snmp_insert_iprteidx_tree
record sysuptime for _this_ route when it is installed (needed for ipRouteAge) in the **netif**.

Global snmp_send_response
do we need separate rx and tx pcbs for threaded case?

Global snmp_send_response
release some memory, retry and return tooBig? tooMuchHassle?

File snmp_structs.h
namespace prefixes

Index