lwIP

Version CVS_HEAD
8/10/2007 12:18 AM

Table of Contents

lwIP Directory Hierarchy

lwIP Directories

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

lwIP Data Structure Index

lwIP Data Structures

Here are the data structures with brief descriptions:

lwIP File Index

lwIP File List

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

lwIP Page Index

lwIP Related Pages

lwIP Directory Documentation

C:/OPENSOURCE/LwIP/src/api/ Directory Reference

Files

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

C:/ Directory Reference

Directories

- directory **OPENSOURCE**

C:/OPENSOURCE/LwIP/src/core/ Directory Reference

Directories

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

Files

- file **dhcp.c**
- file **inet.c**
- file **inet6.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**

C:/OPENSOURCE/LwIP/src/include/ Directory Reference

Directories

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

C:/OPENSOURCE/LwIP/src/include/ipv4/ Directory Reference

Directories

- directory **lwip**

C:/OPENSOURCE/LwIP/src/core/ipv4/ Directory Reference

Files

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

C:/OPENSOURCE/LwIP/src/include/ipv6/ Directory Reference

Directories

- directory **lwip**

C:/OPENSOURCE/LwIP/src/core/ipv6/ Directory Reference

Files

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

C:/OPENSOURCE/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 **err.h**
- file **mem.h**
- file **memp.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**

C:/OPENSOURCE/LwIP/src/include/ipv6/lwip/ Directory Reference

Files

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

C:/OPENSOURCE/LwIP/src/include/ipv4/lwip/ Directory Reference

Files

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

C:/OPENSOURCE/LwIP/ Directory Reference

Directories

- directory **src**

C:/OPENSOURCE/LwIP/src/netif/ Directory Reference

Directories

- directory **ppp**

Files

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

C:/OPENSOURCE/LwIP/src/include/netif/ Directory Reference

Files

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

C:/OPENSOURCE/ Directory Reference

Directories

- directory **LwIP**

C:/OPENSOURCE/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 **pppdebug.h**
- file **randm.c**
- file **randm.h**
- file **vj.c**
- file **vj.h**
- file **vjbsdhdr.h**

C:/OPENSOURCE/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**

C:/OPENSOURCE/LwIP/src/ Directory Reference

Directories

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

lwIP Data Structure Documentation

dhcp_msg Struct Reference

`#include <dhcp.h>`

Detailed Description

minimum set of fields of any DHCP message

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

- C:/OPENSOURCE/LwIP/src/include/lwip/**dhcp.h**

etharp_hdr Struct Reference

`#include <etharp.h>`

Detailed Description

the ARP message

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

- C:/OPENSOURCE/LwIP/src/include/netif/etharp.h

etharp_q_entry Struct Reference

`#include <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:

- C:/OPENSOURCE/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:

- C:/OPENSOURCE/LwIP/src/netif/**ethernetif.c**

mib_array_node Struct Reference

`#include <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:

- C:/OPENSOURCE/LwIP/src/include/lwip/**snmp_structs.h**

mib_external_node Struct Reference

`#include <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

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)

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

async Questions

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

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

number of objects at this level

u8_t mib_external_node::tree_levels

tree levels under this node

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

- C:/OPENSOURCE/LwIP/src/include/lwip/**snmp_structs.h**

mib_list_rootnode Struct Reference

`#include <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:

- C:/OPENSOURCE/LwIP/src/include/lwip/**snmp_structs.h**

mib_node Struct Reference

`#include <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

const u8_t mib_node::node_type

One out of MIB_NODE_AR, MIB_NODE_LR or MIB_NODE_EX

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()**

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

- C:/OPENSOURCE/LwIP/src/include/lwip/**snmp_structs.h**

mib_ram_array_node Struct Reference

`#include <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:

- C:/OPENSOURCE/LwIP/src/include/lwip/**snmp_structs.h**

netif Struct Reference

`#include <netif.h>`

Data Fields

- **netif** * **next**
- **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**
- **dhcp** * **dhcp**
- **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 autoip* netif::autoip

the AutoIP client state information for this netif

struct dhcp* netif::dhcp

the DHCP client state information for this netif

u8_t netif::flags

flags (see NETIF_FLAG_ above)

u8_t netif::hwaddr[NETIF_MAX_HWADDR_LEN]

link level hardware address of this interface

u8_t netif::hwaddr_len

number of bytes used in hwaddr

u32_t netif::ifinoctets

counters

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.

struct ip_addr netif::ip_addr

IP address configuration in network byte order

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

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

u32_t netif::link_speed

(estimate) link speed

u8_t netif::link_type

link type (ifType values per RFC1213)

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.

u16_t netif::mtu

maximum transfer unit (in bytes)

char netif::name[2]

descriptive abbreviation

struct netif* netif::next

pointer to next in linked list

u8_t netif::num

number of this interface

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.

void* netif::state

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

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

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

u32_t netif::ts

timestamp at last change made (up/down)

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

- C:/OPENSOURCE/LwIP/src/include/lwip/netif.h

nse Struct Reference

Data Fields

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

Detailed Description

node stack entry (old news?)

Field Documentation

s32_t nse::r_id

right child identifier

u8_t nse::r_nl

right child next level

struct mib_node* nse::r_ptr

right child

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

- C:/OPENSOURCE/LwIP/src/core/snmp/**mib_structs.c**

obj_def Struct Reference

`#include <snmp_structs.h>`

Detailed Description

object definition returned by (get_object_def)()

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

- C:/OPENSOURCE/LwIP/src/include/lwip/**snmp_structs.h**

snmp_obj_id Struct Reference

`#include <snmp.h>`

Detailed Description

internal object identifier representation

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

- C:/OPENSOURCE/LwIP/src/include/lwip/snmp.h

snmp_resp_header_lengths Struct Reference

`#include <snmp_msg.h>`

Detailed Description

output response message header length fields

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

- C:/OPENSOURCE/LwIP/src/include/lwip/**snmp_msg.h**

snmp_trap_header_lengths Struct Reference

`#include <snmp_msg.h>`

Detailed Description

output response message header length fields

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

- C:/OPENSOURCE/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:

- C:/OPENSOURCE/LwIP/src/core/**sys.c**

lwIP File Documentation

C:/OPENSOURCE/LwIP/src/api/api_lib.c File Reference

```
#include <string.h>
#include "lwip/opt.h"
#include "lwip/api.h"
#include "lwip/api_msg.h"
#include "lwip/tcpip.h"
#include "lwip/memp.h"
```

Functions

- 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)
- netconn * **netconn_new_with_proto_and_callback** (enum netconn_type t, u8_t proto, void(*callback)(struct netconn *, enum netconn_evt, u16_t len))
- err_t **netconn_delete** (struct netconn *conn)
- enum netconn_type **netconn_type** (struct netconn *conn)
- err_t **netconn_peer** (struct netconn *conn, struct ip_addr *addr, u16_t *port)
- err_t **netconn_addr** (struct netconn *conn, struct ip_addr **addr, u16_t *port)
- 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** (struct netconn *conn)
- netconn * **netconn_accept** (struct netconn *conn)
- 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 copy)
- 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)

Detailed Description

Sequential API External module

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)

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

struct netconn* netconn_accept (struct netconn * *conn*)

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_addr (struct netconn * *conn*, struct ip_addr ** *addr*, u16_t * *port*)

Get the local 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 local IP address
port a pointer to which to save the local port (or protocol for RAW)

Returns:

ERR_OK if the information was retrieved

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_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 (struct netconn * *conn*)

Set a TCP netconn into listen mode

Parameters:

conn the tcp netconn to set to listen mode

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*, void(*)(struct netconn *, enum netconn_evt, u16_t len) *callback*)

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

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

Get the current perr a netconn is connected to. This might only be temporary for UDP netconns, doesn't work for RAW netconns and returns garbage if called for a TCP listen netconn.

Parameters:

conn the netconn to query
addr a pointer to which to save the remote IP address
port a pointer to which to save the remote port

Returns:

ERR_CONN for invalid connections ERR_OK if the information was retrieved

struct netbuf* netconn_recv (struct netconn * *conn*)

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 *copy*)

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
copy flag: 1 = copy the data, 0 = data is non-volatile, can be sent by reference

Returns:

ERR_OK if data was sent, any other err_t on error

C:/OPENSOURCE/LwIP/src/api/api_msg.c File Reference

```
#include "lwip/opt.h"
#include "lwip/arch.h"
#include "lwip/api_msg.h"
#include "lwip/memp.h"
#include "lwip/sys.h"
#include "lwip/tcpip.h"
#include "lwip/igmp.h"
```

Functions

- void **do_newconn** (struct api_msg_msg *msg)
- void **do_delconn** (struct api_msg_msg *msg)
- void **do_bind** (struct api_msg_msg *msg)
- void **do_connect** (struct api_msg_msg *msg)
- void **do_disconnect** (struct api_msg_msg *msg)
- void **do_listen** (struct api_msg_msg *msg)
- void **do_send** (struct api_msg_msg *msg)
- void **do_recv** (struct api_msg_msg *msg)
- void **do_write** (struct api_msg_msg *msg)
- void **do_close** (struct api_msg_msg *msg)
- void **do_join_leave_group** (struct api_msg_msg *msg)

Detailed Description

Sequential API Internal module

Function Documentation

void do_bind (struct api_msg_msg * *msg*)

Bind a pcb contained in a netconn Called from netconn_bind.

Parameters:

msg the api_msg_msg pointing to the connection and containing the IP address and port to bind to

void do_close (struct api_msg_msg * *msg*)

Close a TCP pcb contained in a netconn Called from netconn_close

Parameters:

msg the api_msg_msg pointing to the connection

void do_connect (struct api_msg_msg * *msg*)

Connect a pcb contained inside a netconn Called from netconn_connect.

Parameters:

msg the api_msg_msg pointing to the connection and containing the IP address and port to connect to

void do_delconn (struct api_msg_msg * *msg*)

Delete the pcb inside a netconn. Called from netconn_delete.

Parameters:

msg the api_msg_msg pointing to the connection

void do_disconnect (struct api_msg_msg * *msg*)

Connect a pcb contained inside a netconn Only used for UDP netconns. Called from netconn_disconnect.

Parameters:

msg the api_msg_msg pointing to the connection to disconnect

void do_join_leave_group (struct api_msg_msg * *msg*)

Join multicast groups for UDP netconns. Called from netconn_join_leave_group

Parameters:

msg the api_msg_msg pointing to the connection

void do_listen (struct api_msg_msg * *msg*)

Set a TCP pcb contained in a netconn into listen mode Called from netconn_listen.

Parameters:

msg the api_msg_msg pointing to the connection

void do_newconn (struct api_msg_msg * *msg*)

Create a new pcb of a specific type inside a netconn. Called from netconn_new_with_proto_and_callback.

Parameters:

msg the api_msg_msg describing the connection type

void do_recv (struct api_msg_msg * *msg*)

Recv some data from a RAW or UDP pcb contained in a netconn Called from netconn_recv

Parameters:

msg the api_msg_msg pointing to the connection

void do_send (struct api_msg_msg * *msg*)

Send some data on a RAW or UDP pcb contained in a netconn Called from netconn_send

Parameters:

msg the api_msg_msg pointing to the connection

void do_write (struct api_msg_msg * *msg*)

Send some data on a TCP pcb contained in a netconn Called from netconn_write

Parameters:

msg the api_msg_msg pointing to the connection

C:/OPENSOURCE/LwIP/src/api/err.c File Reference

```
#include "lwip/err.h"
```

Detailed Description

Error Management module

C:/OPENSOURCE/LwIP/src/api/netifapi.c File Reference

```
#include "lwip/opt.h"
#include "lwip/arch.h"
#include "lwip/netifapi.h"
#include "lwip/tcpip.h"
```

Functions

- 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_remove** (struct **netif** ***netif**)
- err_t **netifapi_dhcp_start** (struct **netif** ***netif**)
- err_t **netifapi_dhcp_stop** (struct **netif** ***netif**)
- void **do_netifapi_netif_add** (struct netifapi_msg_msg *msg)
- void **do_netifapi_netif_remove** (struct netifapi_msg_msg *msg)
- void **do_netifapi_dhcp_start** (struct netifapi_msg_msg *msg)
- void **do_netifapi_dhcp_stop** (struct netifapi_msg_msg *msg)

Detailed Description

Network Interface Sequential API module

Function Documentation

void do_netifapi_dhcp_start (struct netifapi_msg_msg * *msg*)

TODO

void do_netifapi_dhcp_stop (struct netifapi_msg_msg * *msg*)

TODO

void do_netifapi_netif_add (struct netifapi_msg_msg * *msg*)

TODO

void do_netifapi_netif_remove (struct netifapi_msg_msg * *msg*)

TODO

err_t netifapi_dhcp_start (struct netif * *netif*)

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

Note:

for params

See also:

dhcp_start()

err_t netifapi_dhcp_stop (struct netif * *netif*)

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

Note:

for params

See also:

dhcp_stop()

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_remove (struct netif * *netif*)

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

Note:

for params

See also:

netif_remove()

C:/OPENSOURCE/LwIP/src/api/sockets.c File Reference

```
#include "lwip/sockets.h"
#include <string.h>
#include "lwip/opt.h"
#include "lwip/api.h"
#include "lwip/arch.h"
#include "lwip/sys.h"
#include "lwip/igmp.h"
#include "lwip/inet.h"
#include "lwip/tcp.h"
#include "lwip/tcpip.h"
```

Detailed Description

Sockets BSD-Like API module

C:/OPENSOURCE/LwIP/src/api/tcpip.c File Reference

```
#include "lwip/opt.h"
#include "lwip/sys.h"
#include "lwip/memp.h"
#include "lwip/pbuf.h"
#include "netif/etharp.h"
#include "lwip/ip.h"
#include "lwip/ip_frag.h"
#include "lwip/udp.h"
#include "lwip/tcp.h"
#include "lwip/tcpip.h"
#include "lwip/igmp.h"
```

Functions

- void **tcp_timer_needed** (void)
- err_t **tcpip_callback** (void(*f)(void *ctx), void *ctx)
- err_t **tcpip_apimsg** (struct api_msg *apimsg)
- err_t **tcpip_netifapi** (struct netifapi_msg *netifapimsg)
- void **tcpip_init** (void(*initfunc)(void *), void *arg)

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_callback (void(*)(void *ctx) *f*, void * *ctx*)

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

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 ARP, IP, UDP and TCP
- 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_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

C:/OPENSOURCE/LwIP/src/core/dhcp.c File Reference

```
#include <string.h>
#include "lwip/stats.h"
#include "lwip/mem.h"
#include "lwip/udp.h"
#include "lwip/ip_addr.h"
#include "lwip/netif.h"
#include "lwip/inet.h"
#include "netif/etharp.h"
#include "lwip/sys.h"
#include "lwip/opt.h"
#include "lwip/dhcp.h"
#include "lwip/autoip.h"
```

Functions

- void **dhcp_coarse_tmr** ()
- void **dhcp_fine_tmr** ()
- err_t **dhcp_start** (struct **netif** ***netif**)
- void **dhcp_inform** (struct **netif** ***netif**)
- 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_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

C:/OPENSOURCE/LwIP/src/core/inet.c File Reference

```
#include "lwip/opt.h"
#include "lwip/arch.h"
#include "lwip/def.h"
#include "lwip/inet.h"
#include "lwip/sys.h"
```

Functions

- u16_t **inet_chksum_pbuf** (struct pbuf *p)
- 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 Internet checksum and 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*)

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

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

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

C:/OPENSOURCE/LwIP/src/core/inet6.c File Reference

```
#include "lwip/opt.h"
#include "lwip/def.h"
#include "lwip/inet.h"
```

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

C:/OPENSOURCE/LwIP/src/core/ipv4/autoip.c File Reference

```
#include <stdlib.h>
#include <string.h>
#include "lwip/mem.h"
#include "lwip/udp.h"
#include "lwip/ip_addr.h"
#include "lwip/netif.h"
#include "lwip/autoip.h"
#include "netif/etharp.h"
```

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

C:/OPENSOURCE/LwIP/src/core/ipv4/icmp.c File Reference

```
#include <string.h>
#include "lwip/opt.h"
#include "lwip/icmp.h"
#include "lwip/inet.h"
#include "lwip/ip.h"
#include "lwip/def.h"
#include "lwip/stats.h"
#include "lwip/snmp.h"
```

Functions

- void **icmp_input** (struct pbuf *p, struct **netif** *inp)
- void **icmp_dest_unreach** (struct pbuf *p, enum icmp_dur_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

C:/OPENSOURCE/LwIP/src/core/ipv4/igmp.c File Reference

```
#include "lwip/debug.h"
#include "lwip/def.h"
#include "lwip/mem.h"
#include "lwip/ip.h"
#include "lwip/inet.h"
#include "lwip/netif.h"
#include "lwip/icmp.h"
#include "lwip/udp.h"
#include "lwip/tcp.h"
#include "lwip/stats.h"
#include "lwip/igmp.h"
#include "arch/perf.h"
#include "string.h"
```

Functions

- void **igmp_init** (void)
- igmp_group * **igmp_lookfor_group** (struct **netif** *ifp, struct ip_addr *addr)
- igmp_group * **igmp_lookup_group** (struct **netif** *ifp, struct ip_addr *addr)
- void **igmp_input** (struct pbuf *p, struct **netif** *inp, struct ip_addr *dest)
- err_t **igmp_joingroup** (struct **netif** *ifp, struct ip_addr *groupaddr)
- err_t **igmp_leavegroup** (struct **netif** *ifp, 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)
- 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_init (void)

Initialize this module

Only network interfaces registered when this function is called are igmp-enabled.

This will enable igmp on all interface. In the current implementation it is not possible to have igmp on one interface but not the other.

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

err_t igmp_joingroup (struct netif * *ifp*, struct ip_addr * *groupaddr*)

Join a group on one network interface.

Parameters:

ifp 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, an err_t otherwise

err_t igmp_leavegroup (struct netif * *ifp*, struct ip_addr * *groupaddr*)

Leave a group on one network interface.

Parameters:

ifp 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, an err_t otherwise

struct igmp_group* igmp_lookfor_group (struct netif * *ifp*, struct ip_addr * *addr*)

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*)

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

Parameters:

ifp the network interfacefor which to look
addr the group ip address to search

Returns:

a struct igmp_group*, NULL on memory error.

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

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()**)

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).

C:/OPENSOURCE/LwIP/src/core/ipv4/ip.c File Reference

```
#include "lwip/opt.h"
#include "lwip/def.h"
#include "lwip/mem.h"
#include "lwip/ip.h"
#include "lwip/ip_frag.h"
#include "lwip/inet.h"
#include "lwip/netif.h"
#include "lwip/icmp.h"
#include "lwip/raw.h"
#include "lwip/udp.h"
#include "lwip/tcp.h"
#include "lwip/stats.h"
#include "arch/perf.h"
#include "lwip/snmp.h"
#include "lwip/dhcp.h"
#include "lwip/igmp.h"
```

Functions

- void **ip_init** (void)
- **netif** * **ip_route** (struct ip_addr *dest)
- err_t **ip_input** (struct pbuf *p, struct **netif** *inp)
- err_t **ip_output_if** (struct pbuf *p, struct ip_addr *src, struct ip_addr *dest, u8_t ttl, u8_t tos, u8_t proto, struct **netif** ***netif**)
- err_t **ip_output** (struct pbuf *p, struct ip_addr *src, struct ip_addr *dest, u8_t ttl, u8_t tos, u8_t proto)

Detailed Description

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

See also:

ip_frag.c

Function Documentation

void ip_init (void)

Initializes the IP layer.

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)

err_t ip_output (struct pbuf * *p*, struct ip_addr * *src*, struct ip_addr * *dest*, u8_t *ttl*, u8_t *tos*, u8_t *proto*)

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

See ip_ouptut_if for parameter declaration.

Returns:

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

err_t ip_output_if (struct pbuf * *p*, struct ip_addr * *src*, struct ip_addr * *dest*, u8_t *ttl*, u8_t *tos*, 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. 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"

struct netif* ip_route (struct ip_addr * *dest*)

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

C:/OPENSOURCE/LwIP/src/core/ipv4/ip_addr.c File Reference

```
#include "lwip/ip_addr.h"
#include "lwip/inet.h"
#include "lwip/netif.h"
```

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

C:/OPENSOURCE/LwIP/src/core/ipv4/ip_frag.c File Reference

```
#include <string.h>
#include "lwip/opt.h"
#include "lwip/ip.h"
#include "lwip/ip_frag.h"
#include "lwip/netif.h"
#include "lwip/snmp.h"
#include "lwip/stats.h"
```

Detailed Description

This is the IPv4 packet segmentation and reassembly implementation.

C:/OPENSOURCE/LwIP/src/core/mem.c File Reference

```
#include <string.h>
#include "lwip/arch.h"
#include "lwip/opt.h"
#include "lwip/def.h"
#include "lwip/mem.h"
#include "lwip/sys.h"
#include "lwip/stats.h"
```

Functions

- void * **mem_malloc** (mem_size_t size)

Detailed Description

Dynamic memory manager

Function Documentation

void* mem_malloc (mem_size_t *size*)

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.

Note that the returned value will always be aligned.

C:/OPENSOURCE/LwIP/src/core/memp.c File Reference

```
#include <string.h>
#include "lwip/opt.h"
#include "lwip/memp.h"
#include "lwip/pbuf.h"
#include "lwip/udp.h"
#include "lwip/raw.h"
#include "lwip/tcp.h"
#include "lwip/api.h"
#include "lwip/api_msg.h"
#include "lwip/tcpip.h"
#include "lwip/sys.h"
#include "lwip/stats.h"
#include "netif/etharp.h"
```

Functions

- void **memp_init** (void)
- void * **memp_malloc_fn** (memp_t type, const char *file, const int line)
- void **memp_free** (memp_t type, void *mem)

Detailed Description

Dynamic pool memory manager

Function Documentation

void memp_free (memp_t *type*, void * *mem*)

Put an element back into its pool.

Parameters:

type the pool where to put mem
mem the memp element to free

void memp_init (void)

Initialize this module.

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

void* memp_malloc_fn (memp_t *type*, const char * *file*, const int *line*)

Get an element from a specific pool.

Parameters:

type the pool to get an element from
file file name calling this function
line number of line where this function is called

C:/OPENSOURCE/LwIP/src/core/netif.c File Reference

```
#include "lwip/opt.h"
#include "lwip/def.h"
#include "lwip/ip_addr.h"
#include "lwip/netif.h"
#include "lwip/tcp.h"
#include "lwip/snmp.h"
#include "netif/etharp.h"
```

Functions

- void **netif_init** (void)
- **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**)
- **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**)
- u8_t **netif_is_up** (struct **netif** ***netif**)
- void **netif_set_down** (struct **netif** ***netif**)
- void **netif_set_status_callback** (struct **netif** ***netif**, void(*status_callback)(struct **netif** ***netif**))
- void **netif_set_link_callback** (struct **netif** ***netif**, void(*link_callback)(struct **netif** ***netif**))
- void **netif_set_link_down** (struct **netif** ***netif**)
- void **netif_set_link_up** (struct **netif** ***netif**)

Variables

- **netif** * **netif_list** = NULL
- **netif** * **netif_default** = NULL

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*)

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*)

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')

void netif_init (void)

Initialize this module

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 = NULL

The default network interface.

struct netif* netif_list = NULL

The list of network interfaces.

C:/OPENSOURCE/LwIP/src/core/pbuf.c File Reference

```
#include <string.h>
#include "lwip/opt.h"
#include "lwip/stats.h"
#include "lwip/def.h"
#include "lwip/mem.h"
#include "lwip/memp.h"
#include "lwip/pbuf.h"
#include "lwip/sys.h"
#include "arch/perf.h"
```

Functions

- pbuf * **pbuf_alloc** (pbuf_layer l, u16_t length, pbuf_flag flag)
- 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)
- 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 *l*, u16_t *length*, pbuf_flag *flag*)

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:

l flag to define header size
length size of the pbuf's payload
flag 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*)

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

C:/OPENSOURCE/LwIP/src/core/raw.c File Reference

```
#include <string.h>
#include "lwip/opt.h"
#include "lwip/def.h"
#include "lwip/memp.h"
#include "lwip/inet.h"
#include "lwip/ip_addr.h"
#include "lwip/netif.h"
#include "lwip/raw.h"
#include "lwip/stats.h"
#include "arch/perf.h"
#include "lwip/snmp.h"
```

Functions

- void **raw_init** (void)
- 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)
- 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()**

void raw_init (void)

Initialize this module

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*)

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

C:/OPENSOURCE/LwIP/src/core/snmp/asn1_dec.c File Reference

```
#include "lwip/opt.h"
#include "lwip/snmp_asn1.h"
```

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!!

C:/OPENSOURCE/LwIP/src/core/snmp/asn1_enc.c File Reference

```
#include "lwip/opt.h"
#include "lwip/snmp_asn1.h"
```

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!!

C:/OPENSOURCE/LwIP/src/core/snmp/mib2.c File Reference

```
#include "arch/cc.h"
#include "lwip/opt.h"
#include "lwip/snmp.h"
#include "lwip/netif.h"
#include "netif/etharp.h"
#include "lwip/ip.h"
#include "lwip/ip_frag.h"
#include "lwip/tcp.h"
#include "lwip/udp.h"
#include "lwip/snmp_asn1.h"
#include "lwip/snmp_structs.h"
```

Defines

- #define **SNMP_ENTERPRISE_ID** 26381

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

- **mib_list_rootnode udp_root**
- **mib_list_rootnode tcpconntree_root**
- **mib_list_rootnode ipntomtree_root**
- **mib_list_rootnode iprtetree_root**
- **mib_list_rootnode ipaddrtree_root**
- **mib_list_rootnode arptree_root**
- **mib_list_rootnode iflist_root**
- const 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.

Define Documentation

#define SNMP_ENTERPRISE_ID 26381

IANA assigned enterprise ID for lwIP is 26381

See also:

`http://www.iana.org/assignments/enterprise-numbers`

Note:

this enterprise ID is assigned to the lwIP project, all object identifiers living under this ID are assigned by the lwIP maintainers (contact Christiaan Simons)!
don't change this define, use **snmp_set_sysobjid()**

If you need to create your own private MIB you'll need to apply for your own enterprise ID with IANA: `http://www.iana.org/numbers.html`

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

index root node for ifTable

const struct mib_array_node internet

```
Initial value: {
  &noleafs_get_object_def,
  &noleafs_get_value,
  &noleafs_set_test,
  &noleafs_set_value,
  MIB_NODE_AR,
  1,
  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

C:/OPENSOURCE/LwIP/src/core/snmp/mib_structs.c File Reference

```
#include "lwip/opt.h"
#include "lwip/snmp_structs.h"
#include "lwip/mem.h"
```

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)
- **mib_list_rootnode** * **snmp_mib_node_delete** (struct **mib_list_rootnode** *rn, struct mib_list_node *n)
- **mib_node** * **snmp_search_tree** (struct **mib_node** *node, u8_t ident_len, s32_t *ident, struct snmp_name_ptr *np)
- **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*)

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*)

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*)

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_*()

C:/OPENSOURCE/LwIP/src/core/snmp/msg_in.c File Reference

```
#include "lwip/opt.h"
#include <string.h>
#include "arch/cc.h"
#include "lwip/ip_addr.h"
#include "lwip/mem.h"
#include "lwip/udp.h"
#include "lwip/stats.h"
#include "lwip/snmp.h"
#include "lwip/snmp_asn1.h"
#include "lwip/snmp_msg.h"
#include "lwip/snmp_structs.h"
```

Functions

- void **snmp_init** (void)
- void **snmp_msg_event** (u8_t request_id)
- 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*)

Varbind-list functions.

Variable Documentation

const char snmp_publiccommunity[7] = "public"

default SNMP community string

const s32_t snmp_version = 0

SNMP v1 == 0

C:/OPENSOURCE/LwIP/src/core/snmp/msg_out.c File Reference

```
#include "lwip/opt.h"
#include "arch/cc.h"
#include "lwip/udp.h"
#include "lwip/netif.h"
#include "lwip/snmp.h"
#include "lwip/snmp_asn1.h"
#include "lwip/snmp_msg.h"
```

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

- 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

C:/OPENSOURCE/LwIP/src/core/stats.c File Reference

```
#include <string.h>
#include "lwip/opt.h"
#include "lwip/def.h"
#include "lwip/stats.h"
#include "lwip/mem.h"
```

Detailed Description

Statistics module

C:/OPENSOURCE/LwIP/src/core/sys.c File Reference

```
#include "lwip/sys.h"
#include "lwip/opt.h"
#include "lwip/def.h"
#include "lwip/memp.h"
#include "lwip/tcpip.h"
```

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

C:/OPENSOURCE/LwIP/src/core/tcp.c File Reference

```
#include <string.h>
#include "lwip/opt.h"
#include "lwip/def.h"
#include "lwip/mem.h"
#include "lwip/memp.h"
#include "lwip/snmp.h"
#include "lwip/tcp.h"
```

Functions

- void **tcp_init** (void)
- 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)
- tcp_pcb * **tcp_listen** (struct tcp_pcb *pcb)
- 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)
- tcp_pcb * **tcp_alloc** (u8_t prio)
- tcp_pcb * **tcp_new** (void)
- void **tcp_arg** (struct tcp_pcb *pcb, void *arg)
- 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)

Variables

- tcp_pcb * **tcp_bound_pcbs**
- tcp_listen_pcbs_t **tcp_listen_pcbs**
- tcp_pcb * **tcp_active_pcbs**
- 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

struct tcp_pcb* tcp_alloc (u8_t *prio*)

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_fasttmr (void)

Is called every TCP_FAST_INTERVAL (250 ms) and sends delayed ACKs.

Automatically called from **tcp_tmr()**.

void tcp_init (void)

Initializes the TCP layer.

struct tcp_pcb* tcp_listen (struct tcp_pcb * *pcb*)

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

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)

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.

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!

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_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

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_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

C:/OPENSOURCE/LwIP/src/core/tcp_in.c File Reference

```
#include "lwip/def.h"
#include "lwip/opt.h"
#include "lwip/ip_addr.h"
#include "lwip/netif.h"
#include "lwip/mem.h"
#include "lwip/memp.h"
#include "lwip/inet.h"
#include "lwip/tcp.h"
#include "lwip/stats.h"
#include "arch/perf.h"
#include "lwip/snmp.h"
```

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

C:/OPENSOURCE/LwIP/src/core/tcp_out.c File Reference

```
#include <string.h>
#include "lwip/def.h"
#include "lwip/opt.h"
#include "lwip/mem.h"
#include "lwip/memp.h"
#include "lwip/sys.h"
#include "lwip/ip_addr.h"
#include "lwip/netif.h"
#include "lwip/inet.h"
#include "lwip/tcp.h"
#include "lwip/stats.h"
#include "lwip/snmp.h"
```

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 copy)
- err_t **tcp_enqueue** (struct tcp_pcb *pcb, void *arg, u16_t len, u8_t flags, u8_t copy, 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)

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 *copy*, 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
copy 1 if data must be copied, 0 if data is non-volatile and can be referenced.
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 *copy*)

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
copy 1 if data must be copied, 0 if data is non-volatile and can be referenced.

Returns:

ERR_OK if enqueued, another err_t on error

See also:

tcp_write()

C:/OPENSOURCE/LwIP/src/core/udp.c File Reference

```
#include <string.h>
#include "lwip/opt.h"
#include "lwip/def.h"
#include "lwip/memp.h"
#include "lwip/inet.h"
#include "lwip/ip_addr.h"
#include "lwip/netif.h"
#include "lwip/udp.h"
#include "lwip/icmp.h"
#include "lwip/stats.h"
#include "arch/perf.h"
#include "lwip/snmp.h"
```

Functions

- void **udp_init** (void)
- void **udp_input** (struct pbuf *p, struct **netif** *inp)
- err_t **udp_sendto** (struct udp_pcb *pcb, struct pbuf *p, struct ip_addr *dst_ip, u16_t dst_port)
- err_t **udp_send** (struct udp_pcb *pcb, struct pbuf *p)
- 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)
- udp_pcb * **udp_new** (void)

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

void udp_disconnect (struct udp_pcb * *pcb*)

Disconnect a UDP PCB

Parameters:

pcb the udp pcb to disconnect.

void udp_init (void)

Initialize the UDP module

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)

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()

C:/OPENSOURCE/LwIP/src/include/ipv4/lwip/autoip.h File Reference

```
#include "lwip/opt.h"
#include "lwip/netif.h"
#include "lwip/udp.h"
#include "netif/etharp.h"
```

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.

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

C:/OPENSOURCE/LwIP/src/include/lwip/dhcp.h File Reference

```
#include "lwip/opt.h"
#include "lwip/netif.h"
#include "lwip/udp.h"
```

Defines

- #define **DHCP_COARSE_TIMER_SECS** 60
- #define **DHCP_FINE_TIMER_MSECS** 500
- #define **DHCP_OPTIONS_LEN** DHCP_MIN_OPTIONS_LEN
- #define **DHCP_MSG_OFS** (UDP_DATA_OFS)
- #define **DHCP_REQUESTING** 1
- #define **DHCP_BACKING_OFF** 12
- #define **DHCP_AUTOIP_COOP_STATE_OFF** 0
- #define **DHCP_OPTION_PAD** 0
- #define **DHCP_OPTION_REQUESTED_IP** 50
- #define **DHCP_OVERLOAD_NONE** 0

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_coarse_tmr** (void)
- void **dhcp_fine_tmr** (void)

Variables

- PACK_STRUCT_BEGIN struct **dhcp_msg PACK_STRUCT_STRUCT**

Detailed Description

Define Documentation

#define DHCP_AUTOIP_COOP_STATE_OFF 0

AUTOIP cooperatation flags

#define DHCP_BACKING_OFF 12

not yet implemented #define DHCP_RELEASING 11

#define DHCP_COARSE_TIMER_SECS 60

period (in seconds) of the application calling **dhcp_coarse_tmr()**

#define DHCP_FINE_TIMER_MSECS 500

period (in milliseconds) of the application calling **dhcp_fine_tmr()**

#define DHCP_MSG_OFS (UDP_DATA_OFS)

DHCP message item offsets and length

#define DHCP_OPTION_PAD 0

BootP options

#define DHCP_OPTION_REQUESTED_IP 50

DHCP options

#define DHCP_OPTIONS_LEN DHCP_MIN_OPTIONS_LEN

set this to be sufficient for your options in outgoing DHCP msgs

#define DHCP_OVERLOAD_NONE 0

possible combinations of overloading the file and sname fields with options

#define DHCP_REQUESTING 1

DHCP client states

Function Documentation

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

PACK_STRUCT_END 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

Variable Documentation

PACK_STRUCT_BEGIN struct dhcp_msg PACK_STRUCT_STRUCT

minimum set of fields of any DHCP message

C:/OPENSOURCE/LwIP/src/include/lwip/snmp_asn1.h File Reference

```
#include "lwip/opt.h"
#include "arch/cc.h"
#include "lwip/err.h"
#include "lwip/pbuf.h"
#include "lwip/snmp.h"
```

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!!

C:/OPENSOURCE/LwIP/src/include/lwip/snmp_msg.h File Reference

```
#include "lwip/opt.h"
#include "arch/cc.h"
#include "lwip/snmp.h"
#include "lwip/snmp_structs.h"
```

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)
- 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]
- snmp_msg_trap **trap_msg**

Detailed Description

SNMP Agent message handling structures.

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)

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*)

Varbind-list functions.

Variable Documentation

const char snmp_publiccommunity[7]

default SNMP community string

const s32_t snmp_version

SNMP v1 == 0

struct snmp_msg_trap trap_msg

TRAP message structure

C:/OPENSOURCE/LwIP/src/include/lwip/snmp_structs.h File Reference

```
#include "lwip/opt.h"
#include "arch/cc.h"
#include "lwip/snmp.h"
```

Defines

- #define **MIB_NODE_SC** 0x01
- #define **MIB_NODE_AR** 0x02
- #define **MIB_NODE_RA** 0x03
- #define **MIB_NODE_LR** 0x04
- #define **MIB_NODE_EX** 0x05

Typedefs

- typedef **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)
- **mib_list_rootnode** * **snmp_mib_node_delete** (struct **mib_list_rootnode** *rn, struct mib_list_node *n)
- **mib_node** * **snmp_search_tree** (struct **mib_node** *node, u8_t ident_len, s32_t *ident, struct snmp_name_ptr *np)
- **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 struct **mib_array_node internet**

Detailed Description

Generic MIB tree structures.

Todo:

namespace prefixes

Define Documentation

#define MIB_NODE_AR 0x02

MIB const array node

#define MIB_NODE_EX 0x05

MIB node for external objects

#define MIB_NODE_LR 0x04

MIB list root node (mem_malloced from RAM)

#define MIB_NODE_RA 0x03

MIB array node (mem_malloced from RAM)

#define MIB_NODE_SC 0x01

MIB const scalar (.0) node

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*)

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*)

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*)

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 struct mib_array_node internet

export MIB tree from **mib2.c**

C:/OPENSOURCE/LwIP/src/netif/etharp.c File Reference

```
#include <string.h>
#include "lwip/opt.h"
#include "lwip/inet.h"
#include "netif/etharp.h"
#include "lwip/ip.h"
#include "lwip/stats.h"
#include "lwip/snmp.h"
#include "lwip/autoip.h"
```

Defines

- #define **ARP_MAXAGE** 240
- #define **ARP_MAXPENDING** 2
- #define **ETHARP_TRY_HARD** 1

Functions

- void **etharp_init** (void)
- void **etharp_tmr** (void)
- s8_t **etharp_find_addr** (struct **netif** ***netif**, struct ip_addr *ipaddr, struct eth_addr **eth_ret, struct ip_addr **ip_ret)
- void **etharp_ip_input** (struct **netif** ***netif**, struct pbuf *p)
- void **etharp_arp_input** (struct **netif** ***netif**, struct eth_addr *ethaddr, struct pbuf *p)
- err_t **etharp_output** (struct **netif** ***netif**, struct pbuf *q, struct ip_addr *ipaddr)
- err_t **etharp_query** (struct **netif** ***netif**, struct ip_addr *ipaddr, struct pbuf *q)
- err_t **etharp_raw** (struct **netif** ***netif**, const struct eth_addr *ethsrc_addr, const struct eth_addr *ethdst_addr, const struct eth_addr *hwsrc_addr, const struct ip_addr *ipsrc_addr, const struct eth_addr *hwdst_addr, const struct ip_addr *ipdst_addr, const u16_t opcode)
- err_t **etharp_request** (struct **netif** ***netif**, struct ip_addr *ipaddr)

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.

Define Documentation

#define ARP_MAXAGE 240

the time an ARP entry stays valid after its last update, for ARP_TMR_INTERVAL = 5000, this is (240 * 5) seconds = 20 minutes.

#define ARP_MAXPENDING 2

the time an ARP entry stays pending after first request, for ARP_TMR_INTERVAL = 5000, this is (2 * 5) seconds = 10 seconds.

#define ETHARP_TRY_HARD 1

Try hard to create a new entry - we want the IP address to appear in the cache (even if this means removing an active entry or so).

Function Documentation

void etharp_arp_input (struct netif * *netif*, struct eth_addr * *ethaddr*, struct pbuf * *p*)

Responds to ARP requests to us. Upon ARP replies to us, add entry to cache send out queued IP packets. Updates cache with snooped address pairs.

Should be called for incoming ARP packets. The pbuf in the argument is freed by this function.

Parameters:

netif The lwIP network interface on which the ARP packet pbuf arrived.
ethaddr Ethernet address of netif.
p The ARP packet that arrived on netif. Is freed by this function.

Returns:

NULL

See also:

pbuf_free()

s8_t etharp_find_addr (struct netif * *netif*, struct ip_addr * *ipaddr*, struct eth_addr ** *eth_ret*, struct ip_addr ** *ip_ret*)

Finds (stable) ethernet/IP address pair from ARP table using interface and IP address index.

Note:

the addresses in the ARP table are in network order!

Parameters:

netif points to interface index
ipaddr points to the (network order) IP address index
eth_ret points to return pointer
ip_ret points to return pointer

Returns:

table index if found, -1 otherwise

void etharp_init (void)

Initializes ARP module.

void etharp_ip_input (struct netif * *netif*, struct pbuf * *p*)

Updates the ARP table using the given IP packet.

Uses the incoming IP packet's source address to update the ARP cache for the local network. The function does not alter or free the packet. This function must be called before the packet p is passed to the IP layer.

Parameters:

netif The lwIP network interface on which the IP packet pbuf arrived.
p The IP packet that arrived on netif.

Returns:

NULL

See also:

pbuf_free()

err_t etharp_output (struct netif * *netif*, struct pbuf * *q*, struct ip_addr * *ipaddr*)

Resolve and fill-in Ethernet address header for outgoing packet.

For IP multicast and broadcast, corresponding Ethernet addresses are selected and the packet is transmitted on the link.

For unicast addresses, the packet is submitted to **etharp_query()**. In case the IP address is outside the local network, the IP address of the gateway is used.

Parameters:

netif The lwIP network interface which the IP packet will be sent on.
q The pbuf(s) containing the IP packet to be sent.
ipaddr The IP address of the packet destination.

Returns:

- ERR_RTE No route to destination (no gateway to external networks), or the return type of either **etharp_query()** or etharp_send_ip().

err_t etharp_query (struct netif * *netif*, struct ip_addr * *ipaddr*, struct pbuf * *q*)

Send an ARP request for the given IP address and/or queue a packet.

If the IP address was not yet in the cache, a pending ARP cache entry is added and an ARP request is sent for the given address. The packet is queued on this entry.

If the IP address was already pending in the cache, a new ARP request is sent for the given address. The packet is queued on this entry.

If the IP address was already stable in the cache, and a packet is given, it is directly sent and no ARP request is sent out.

If the IP address was already stable in the cache, and no packet is given, an ARP request is sent out.

Parameters:

netif The lwIP network interface on which ipaddr must be queried for.
ipaddr The IP address to be resolved.
q If non-NULL, a pbuf that must be delivered to the IP address. q is not freed by this function.

Note:

q must only be ONE packet, not a packet queue!

Returns:

- ERR_BUF Could not make room for Ethernet header.
- ERR_MEM Hardware address unknown, and no more ARP entries available to query for address or queue the packet.
- ERR_MEM Could not queue packet due to memory shortage.
- ERR_RTE No route to destination (no gateway to external networks).
- ERR_ARG Non-unicast address given, those will not appear in ARP cache.

err_t etharp_raw (struct netif * *netif*, const struct eth_addr * *ethsrc_addr*, const struct eth_addr * *ethdst_addr*, const struct eth_addr * *hwsrc_addr*, const struct ip_addr * *ipsrc_addr*, const struct eth_addr * *hwdst_addr*, const struct ip_addr * *ipdst_addr*, const u16_t *opcode*)

Send a raw ARP packet (opcode and all addresses can be modified)

Parameters:

netif the lwip network interface on which to send the ARP packet
ethsrc_addr the source MAC address for the ethernet header
ethdst_addr the destination MAC address for the ethernet header
hwsrc_addr the source MAC address for the ARP protocol header
ipsrc_addr the source IP address for the ARP protocol header
hwdst_addr the destination MAC address for the ARP protocol header
ipdst_addr the destination IP address for the ARP protocol header
opcode the type of the ARP packet

Returns:

ERR_OK if the ARP packet has been sent ERR_MEM if the ARP packet couldn't be allocated any other err_t on failure

err_t etharp_request (struct netif * *netif*, struct ip_addr * *ipaddr*)

Send an ARP request packet asking for ipaddr.

Parameters:

netif the lwip network interface on which to send the request
ipaddr the IP address for which to ask

Returns:

ERR_OK if the request has been sent ERR_MEM if the ARP packet couldn't be allocated any other err_t on failure

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.

C:/OPENSOURCE/LwIP/src/netif/ethernetif.c File Reference

```
#include "lwip/opt.h"
#include "lwip/def.h"
#include "lwip/mem.h"
#include "lwip/pbuf.h"
#include "lwip/sys.h"
#include <lwip/stats.h>
#include "netif/etharp.h"
```

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

C:/OPENSOURCE/LwIP/src/netif/loopif.c File Reference

```
#include "lwip/opt.h"
```

Detailed Description

Loop Interface

C:/OPENSOURCE/LwIP/src/netif/slipif.c File Reference

```
#include "netif/slipif.h"
#include "lwip/opt.h"
#include "lwip/def.h"
#include "lwip/pbuf.h"
#include "lwip/sys.h"
#include "lwip/stats.h"
#include "lwip/sio.h"
```

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 Page Documentation

Todo List

File asn1_dec.c

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

Global snmp_asn1_dec_length (struct pbuf *p, u16_t ofs, u8_t *octets_used, u16_t *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 (u8_t dflt, struct netif *ni)

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

Global snmp_send_response (struct snmp_msg_pstat *m_stat)

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

Global snmp_send_response (struct snmp_msg_pstat *m_stat)

release some memory, retry and return tooBig? tooMuchHassle?

File snmp_structs.h

namespace prefixes

Index