lwIP

Version CVS_HEAD
7/10/2007 9:25 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

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

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 * **state**
- u8_t **hwaddr_len**
- u8_t **hwaddr** [NETIF_MAX_HWADDR_LEN]
- u16_t **mtu**
- u8_t **flags**
- char **name** [2]
- u8_t **num**

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

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

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

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.

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

- C:/OPENSOURCE/LwIP/src/include/lwip/netif.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/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"
```

Detailed Description

Dynamic Host Configuration Protocol client

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"
```

Detailed Description

AutoIP Automatic LinkLocal IP Configuration

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/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"
```

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**)

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_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_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()

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

- void **pbuf_init** (void)
- 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)

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

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_init (void)

Initializes the pbuf module.

Do some checks an initialize the pbuf pool.

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"
```

Detailed Description

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

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

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

Detailed Description

Abstract Syntax Notation One (ISO 8824, 8825) decoding

Todo:

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

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

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

Detailed Description

Abstract Syntax Notation One (ISO 8824, 8825) encoding

Todo:

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

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

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

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.

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

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

Detailed Description

MIB tree access/construction functions.

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

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

Detailed Description

SNMP input message processing (RFC1157).

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

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

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.

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"
```

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.

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"
```

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).

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"
```

Detailed Description

Transmission Control Protocol, outgoing traffic

The output functions of TCP.

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"
```

Detailed Description

User Datagram Protocol module

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

void autoip_init (void)

Init srand, has to be called before entering mainloop

err_t autoip_start (struct netif * *netif*)

Start AutoIP client

err_t autoip_stop (struct netif * *netif*)

Stop 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)

to be called every minute

void dhcp_fine_tmr (void)

to be called every half second

void dhcp_inform (struct netif * *netif*)

inform server of our manual IP address

err_t dhcp_release (struct netif * *netif*)

release the DHCP lease, usually called before **dhcp_stop()**

err_t dhcp_renew (struct netif * *netif*)

enforce early lease renewal (not needed normally)

PACK_STRUCT_END err_t dhcp_start (struct netif * *netif*)

start DHCP configuration

void dhcp_stop (struct netif * *netif*)

stop DHCP configuration

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"
```

Detailed Description

Abstract Syntax Notation One (ISO 8824, 8825) codec.

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

Variables

- const s32_t **snmp_version**
- const char **snmp_publiccommunity** [7]

Detailed Description

SNMP Agent message handling structures.

Function Documentation

void snmp_init (void)

Agent setup, start listening to port 161.

void snmp_msg_event (u8_t *request_id*)

Handle an internal (recv) or external (private response) event.

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]

Agent default "public" community string

const s32_t snmp_version

Agent Version constant, 0 = v1 oddity

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

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

Detailed Description

Generic MIB tree structures.

Todo:

namespace prefixes

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"
```

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_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_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.

lwIP Page Documentation

Todo List

File asn1_dec.c

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

File asn1_enc.c

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

File snmp_structs.h

namespace prefixes

Index