ITRON TCP/IP API Specification Based Protocol Stack

ecTCP/IP

User's Guide

Table of Contents

Chapter 1 Introduction

1.1 Introduction

In ecRTOS, TCP/IP Protocol stack (ecTCP/IP) is added to ecRTOS Kernel for conformity to µT-kernel 3.0 specification and IEEE 2050-2018 standard of Real Time OS. Since only the TCP/IP part is explained in this book, please refer to the ecRTOS User Guide (Kernel Edition) for the OS Part.

1.2 Features

ecRTOS based on µITRON specification for OS, has built-in standard TCP/IP protocol stack. Including first time users of TCP/IP in Embedded Systems, many users have adopted it with satisfactory results.

ecRTOS supports many processors like ARM, PowerPC, NiosV, RISC-V, RX etc. The introduction of CPU of the same family is easy, because the User License system that is similar to compiler License, can be used without any restriction of CPU number or application model. The royalty payment is not related to the number of products. This reduces the user's production cost.

Based on ITRON TCP/IP API specifications suitable for scarce resources of Embedded Systems, it is made as a full-scale protocol stack supporting TCP, UDP, IP, ARP, ICMP and IGMP. Moreover, "Sliding Window" method is used for speed control. Speed and high reliability are achieved by using Slow-start, Fast retransmit and Fast Recivery methods for congestion control.

ecTCP/IP module is provided as separate library for each compiler/version. Besides, complete source code of the library is provided, so that it is possible to perform debug operation inside the protocol stack.

Sample programs of server/client for Telent, FTP and TFTP each, DHCP client and DNS resolver are provided as application layer. It is also possible to provide sample driver for Ethernet controllers like SMSC LAN91C111, ASIX AX88796L, AMD Am79C973, REALTEK RTL8019AS, CIRRUS LOGIC CS8900A, NS DP83902A, Fujitsu MB86964 and processor built-in controllers.

Dual stack of IPv6/IPv4 (ecTCP/IPv6) is provided as an option.

1.3 Limitations

Operation cannot be guaranteed if ported to OS other than ecRTOS.

1.4 File Composition

File Configuration of ecTCP/IP is as follows:

ecRTOS/INC/

ecnet.h	ecTCP/IP Standard Header
ecnetc.h	ecTCP/IP Configuration Header
ecnets.h	ecTCP/IP Internal Definition Header
ecnitod.h	Header for number-charater string conversion functions

ecRTOS/LIB/(CPU)/(CCC)/

ecnxxx.lib	ecTCP/IP Library
ecdxxx.lib	ecTCP/IP Library (With debug information)

ecRTOS/SRC/

ecnet.c	Source of ecTCP/IP service call
ecntcp1.c	Source 1/2 of TCP Module
ecntcp2.c	Source 2/2 of TCP Module
ecneudp.c	Source of UDP module
ecnicmp.c	Source of ICMP module
ecnearp.c	Source of ARP Source
ecnetip.c	Source of IP module
ecnigmp.c	Source of IGMP module
ecneipf.c	Source of IP fragmentation module
ecnelan.c	Source of Networ driver Interface
ecnitod.c	Source of number-character string conversion functions
ecnslib.c	Source of function unavailable in C Standard Library

Folder name (CPU) is made by abbreviated CPU name; (CCC) is made by abbreviated compiler name. Filename xxx corresponds to processor type. Depending on the processing system, an extension may not be lib. ecRTOS/SRC folder contains the source that is compiled and added to the libarry (ecnxxx.lib and ecdxxx.lib). Therefore, user does not usually need to compile/link each of the source files.

ecnet.h

Please include ecnet.h in all the source files that use the functions of ecTCP/IP. This header includes protoype declaration of all ecTCP/IP service calls and definition of structures and constants required for service calls.

ecnetc.h

Please incluce ecnetc.h in only one source file of the application. In this header, internal management block of the protocol stack is defined. Before using #include ecnetc.h, the configuration of ecTCP/IP can be performed by using #define to describe maximum number of Communication End Points or other various parameters.

ecnets.h

ecnetc.h is included by all the source files in the folder ecRTOS¥SRC; hence it is not necessary to include this file directly from the application. In this header, internal structures and contants used by the protocol stack are defined.

ecnitod.h

In this header, number/character sequence conversion functions, not available in C standard library, are declared.

ecnxxx.lib, ecdxxx.lib

Please link the library ecnxxx.lib from the folder ecRTOS¥LIB¥(CPU)¥(CCC), with the user program. Otherwise, in order to debug inside the protocol stack, please link ecdxxx.lib library. All the functions of ecTCP/IP are contained in these libraries, except the LAN driver and samples of application layer programs.

ecnet.c

This contains the common functions called from the following source and the functions not contiained in others sources.

ecntcp1.c, ecntcp2.c

In these sources, control part of TCP (Transmission Control Protocol) is described.

ecneudp.c

In this source, control part of UDP (User Datagram Protocol) is described.

ecnearp.c

In this source, control part of ARP (Address Resolution Protocol) is described.

ecnicmp.c

In this source, control part of ICMP (Internet Control Message Protocol) is described. Since only basic Echo process is performed, user should expand the functionality.

ecnetip.c

In this source, the control part of the most basic and fundamental portion of protocol stack, IP (Internet Protocol) is described.

ecnigmp.c

In this source, control part of IGMP (Internet Global Management Protocol) is described.

ecneipf.c

In this source, control part of IP fragmentation is described.

ecnitod.c

In this source, number/character sequence conversion functions, not available in C standard library, are described.

ecnslib.c

In this source, the functions not present in the C standard library, are described.

1.5 Terminology

Communication End Point

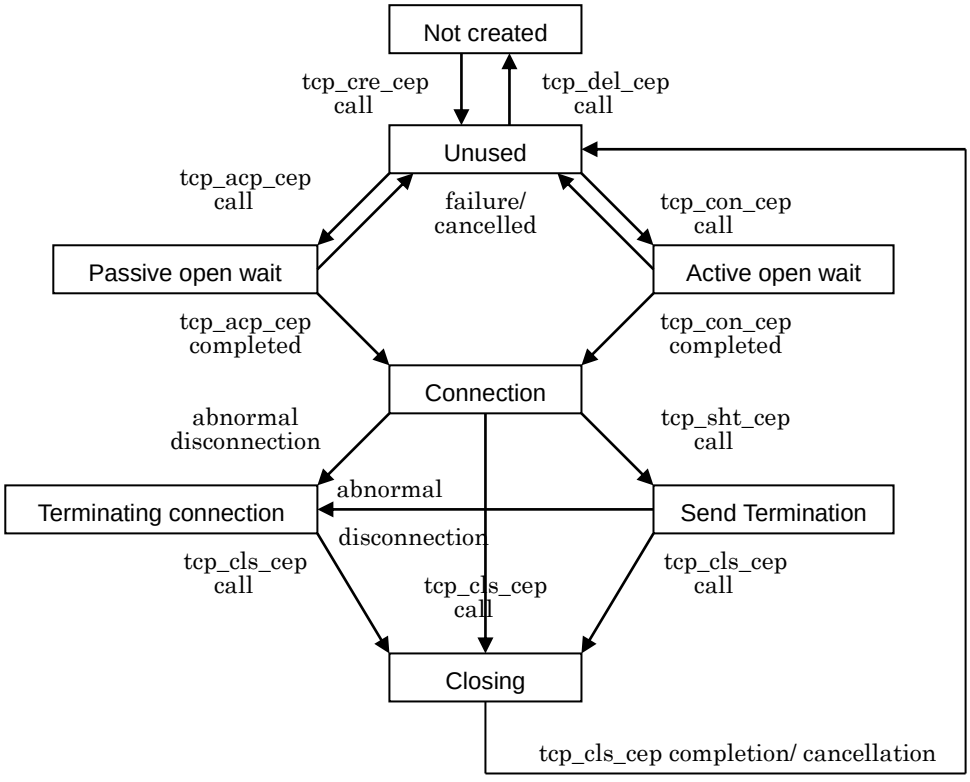
Two types of TCP objects (for operation os service call) are used, TCP Reception Point and TCP Communication End Point. TCP Reception Point is used with TCP Communication End Point in passive mode when waiting for the connection request from the remote node.

As object for UDP, UDP Communication End Point is used.

While being used in the name of service call or as a parameter, TCP Reception Point is abbreviated as rep. TCP Communication End Point and UDP Communication End Point are abbreviated as cep. Each Communication End Point is distinguished by an ID number, beginning from 1.

State of TCP Communication End Point

TCP Communication End Point changes 8 states, Not Created ~ Closed, as follows:



In the 7 states, except “Not Created”, it is called “Created” and in the 6 states, except “Not Created” and “Not Used”, it is called “Used”.

Moreover, in 5 states, except “Connection”, “Send Termination” and “Connection disconnect”, it is called “Not Connected”.

Service Call

The function called from application and defined in ITRON TCP/IP API specification is called Service Call.

Timeout

Timeout function is used by the service call when it is possible for the calling task to be in Wait state. If the time specified as timeout passes when the service call is not complete, internal processing is stopped and error is returned from the service call.

Non blocking

A non blocking function is also prepared for the service call whose calling task cannot be in Wait state. By specifying nonblocking, although the service call returns immediately, internal processing of protocol stack is continued.

Because the calling task cannot know the completion of processing by using service call with nonblocking specification (nonblocking call), following callback function is prepared.

Callback

Callback is a function, specified by the application (Callback routine) and called by the system side (In this case, protocol stack).

Completion of the process, started by the non-blocking call, is notified using callback. The event of UDP packet reception is also notified using callback.

Reduced copy API

In the service call of TCP, an API (Reduced Copy API) is supported to reduce the number of copying of transmitted/received data within the protocol stack. By using “Reduced Copy API”, the application reads and writes directly to the memory pool managed inside the protocol stack and copying of data is reduced by one.

Packet

Although, a block of data in a network is called a packet or datagram (Segment in TCP and for frame in Ethernet), it is called a packet in this document. The UDP packet is same as UDP datagram. TCP packet is same as TCP segment or TCP datagram. Ethernet packet is same as Ethernet frame.

Chapter 2 Composition of Protocol Stack

The composition and operation of ecTCP/IP Protocol Stack is explained in this chapter, according to the following modules.

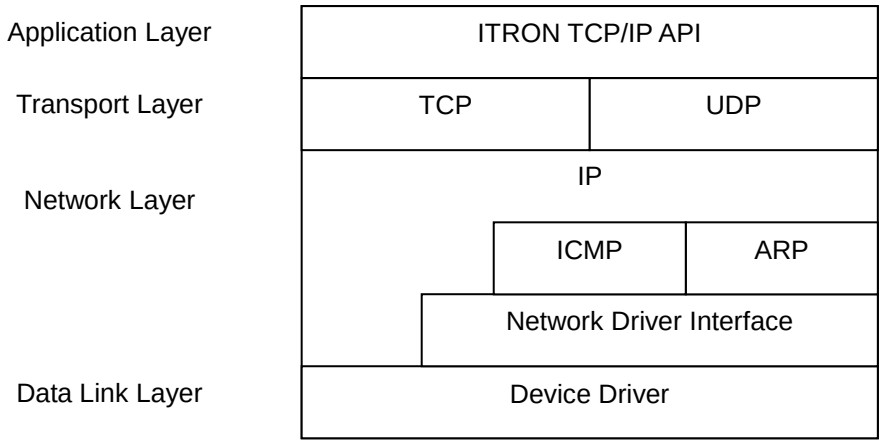
- Network Driver Interface
- Device Driver
- IP Module
- ARP Module
- ICMP Module
- UDP Module
- TCP Module

The contents explained here are not described in ITRON TCP/IP API specifications and are original to ecTCP/IP. Especially, the device driver interface. ARP and ICMP are not as per ITRON TCP/IP specification.

2.1 Outline

Hierarchy Structure

ecTCP/IP Protocol stack consists of following modules:



Protocol Control Task

ecTCP/IP Protocol Stack consists of following 2 tasks and 1 cyclic handler.

IP Receive Task	**1 at all the time**
IP Send Task	**1 at all the time**
TCP/IP Timer (Cyclic Handler)	**1 at all the time**

TCP, UDP, IP, ICMP, IGMP, ARP, Network Driver Interface and all the device drivers are managed by IP Receive Task/IP Send Task and TCP Timer. That is, there is no separate task or cyclic handler of each module.

Memory Pool of the Protocol Stack

The following memory pools are used inside the protocol stack. These are fixed-length pools and the number of memory blocks to be used is configurable.

Memory pool for Ethernet Packets	**1 at all the time**
Memory pool for UDP header	**For each UDP Communication End Point**

The area of memory pool for Ethernet packets is used for transmitting and receiving Ethernet packets. There is only one memory pool at all times and it is shared for transmission and reception of all protocols except UDP transmission. The default number of memory blocks acquired from this memory pool is 16. The size of each memory block is fixed at 1560 bytes to store header and data of all protocols. It is only used not only for transmission and reception of Ethernet packet in a data link layer but for the analysis of packet at every stage and processing of reassembly. Hence, there is no need to acquire new memory when data is transmitted between protocol layers.

The case for UDP transmission is different. The memory pool for UDP header is prepared independently by making the data domain specified by application, as the domain for Ethernet packet. There is a separate memory pool for every UDP Communication End Point. The default number of memory blocks of size 80 bytes each is 2.

Mailbox of Protocol Stack

The following mail boxes are used inside the protocol stack. The communication speed is increased by using a mailbox, since only the pointer of a block is passed and copy of data does not occur.

Send packet queue	**1 at all the time**
Send retry queue	**1 at all the time**
UDP reception queue	**one for each UDP Communication End Point**

Once packet transmitted from each module is queued in the mail box used as Send packet queue, it is passed to data link layer sequentially. Besides, the mailbox can be used as Transmission Retry Queue for packets to resend or pending for transmission.

In case of UDP reception, since received data is directly copied to the application specified receive buffer, UDP receive buffer is used as mailbox for queuing. There is a separate mailbox of every UDP Communication End Point.

Timeout and cancel

Timout is observed at the task level of the application that has issued the service call. When a timeout occurred, the queued packet is searched and removed. It is same in the case of service call for cancellation of pending process or deletion of Communication End Point.

In ecTCP/IP, timout can be set with following parameters:

No Timeout (tmout = TMO_FEVR)
Timeout (tmout = 1~0x7fffffff)
Polling (tmout = TMO_POL)
Non blocking (tmout = TMO_NBLK)

If no timeout is used (tmout = TMO_FEVR) and the connection goes down, service call may not return for a long time or permanently. Hence, please use these service calls with timeout as parameter.

2.2 Network Driver Interface

Composition

Network Driver Interface is the function group that abstracts the LAN controller and is the module between the device driver and the Network Layer. In Network Layer, except for the functions related to Send/Receive Wait, device driver functions should not be called directly and network driver interface should be used. If network driver interface is customized, it can respond to devices other than Ethernet.

Usually, the data link layer sets up the Ethernet header of a transmitting packet (MAC address etc.); in ecTCP/IP protocol stack, it is done in IP layer, to simplify the design of device driver. Moreover, for the improvement in the performance, entire data of the received packet is not read, the interface reads only necessary minimum data.

Obtain length of received packet

[Form] **UH lan_received_len(void)**

[Return Value] **The received number of bytes**

The size of entire packet, received by the device driver, is returned. It should be called before calling lan_read_pkt function.

Receive packet read-out

[Form] **BOOL lan_read_pkt(void *buf, int len)**

	buf	**buffer for reading data**
	len	**number of byte to read**

[Return Value]	**TRUE**	**Normal end**
	FALSE	**Error**

Specified number of bytes is read from the buffer memory of the device. To read a complete packet, this function may be called repeatedly (The entire data may not be read at once).

Receive packet read-out till end

[Form] **BOOL lan_read_pkt_end(void *buf, int len, int bufsz)**

	buf	**Buffer for reading data**
	len	**Number of bytes to read**
	bufsz	**size of buffer**

[Return Value]	**TRUE**	**Normal end**
	FALSE	**Error**

If bufsz of smaller than the size of data received by the driver, read-out of data equal to bufsz is

done and remaining data is ignored. If bufsz is bigger than the received data size, the entire received packet is read. This function is called after calling lan_rad_pkt function, to read all the remaining data.

Cancellation of receive packet

[Form] **void lan_ignore_pkt(void)**

[Return Value] **None**

The packet remaining, after reading data using lan_read_pkt function, is cancelled. When there is no need to read data anymore from a received packet, this function is called instead of lan_read_pkt and lan_read_pkt_end functions.

Writing of packet to send

[Form] **BOOL lan_write_pkt(const void *head, int hlen, const void *data, int dlen)**

head	**First half of the data to write**
hlen	**Number of bytes of the first half of data**
data	**Second half of the data to write**
dlen	**Number of bytes of the second half of data**

[Return Value]	**TRUE**	**Normal end**
	FALSE	**Error**

Data to be transmitted is written to the buffer memory of the device. Since header and data parts are stored separately in UDP transmission, head/hlen and data/dlen are separate, otherwise, only head and hlen are used with data= NULL and dlen=0.

Error processing of LAN driver

[Form] **ER lan_error(ER ercd)**

[Return Value] **Error code**

This function is called when an error occurs in the device driver. In the standard implementation, there is no processing for this.

2.3 Device driver

Composition

There is no separate task in the standard LAN driver for ecTCP/IP. Device driver, the function group called from Network Driver Interface, consists of interrupt handlers that wake up IP Receive Task/IP Send Task. In device driver, packets are transmitted and received irrespective of the protocol type.

Only lan_wai_snd and lan_wai_rcv functions are directly called from IP layer. All the other functions are called from network driver interface. It is not necessary to define the callback functions of device driver.

Initialization of device

[Form] **ER lan_ini(UB *macaddr)**

macaddr **MAC Address (Array of 6 bytes)**

[Return Value] **Error Code**

This function is called from TCP/IP initialization function tcp_ini, to define interrupt handler and set initial values in device registers.

Interrupt handler

[Form] **INTHDR lan_int(void)**

In reception interrupt, this function releases the IP Receive Task waiting for completion of reception in lan_wai_rcv function. In transmission interrupt, this function releases the IP Send Task waiting for completion of transmission in lan_wai_snd function.

Any function name other than lan_int can be used to define an interrupt handler for device driver itself. It can also be executed as interrupt service routine.

Wait for Receive Interrupt

[Form] **ER lan_wai_rcv(TMO tmout)**

tmout **Timeout value**

[Return Value] **Error code**

When there are no received packets in the buffer memory of the device, the IP Receive Task that has called this function, waits for the occurance of receive interrupt. This wait of IP Receive Task is released by the interrupt handler. Moreover, it is called from IP Receive Task with no timeout (tmout = TMO_FEVR), timeout function is not used in present implementation.

Wait for transmission Interrupt

[Form] **ER lan_wai_snd(TMO tmout)**

tmout timeout value

[Return Value] Error value

When there is no space in the buffer memory of the device and a packet cannot be transmitted, the IP Send Task that has called this function waits for the transmission interrupt. This wait of IP Send Task is released by the interrupt handler. Moreover, it is called from IP Send Task with no timeout (tmout = TMO_FEVR), the timeout function is not used in present condition.

Obtaining length of received packet

[Form] ER lan_get_len(UH *len)

len number of bytes of received packet

[Return Value] Error code

This function is called from the lan_received_len function of the network driver interface. This function returns the length of data that should be read from the buffer memory of the device.

Extract received packet

[Form] ER lan_get_pkt(void *buf, int len)

buf buffer for reading

len number of bytes to read

[Return Value] Error code

This function is called from lan_read_pkt function of Netowork Driver Interface. The data of the specified size is read from the buffer memory of the device. This function may be called many times to read one packet (Entire data may not be read at once). The address of buf parameter must be in the multiple of 2 (WORD boundry). When lan_read_pkt function is called repeatedly, the value of len is always even, except for the last time.

When lan_get_pkt function is called, depending on device, it can read receiving data or it has to read all the data together at once. In the latter case, where entire data of received packet is read together when lan_get_pkt function is called, device driver must have a receive buffer to store the data until the lan_get_pkt is called.

Extract received packet till end

[Form] ER lan_get_end(void)

[Return Value] Error Code

This function is called exactly after the lan_get_pkt function is called (repeatedly) from lan_read_pkt function of the Network Driver Interface and the last data has been read. In the device driver, if required, the process of extracting the received packet is completed that is it

prepares to read the following received packet.

Skip reading in received packet

[Form] **ER lan_skp_pkt(int len)**

len **Number of byters to skip**

[Return Value] **Error Code**

This function is called from lan_ignore_pkt function of the Network Driver Interface. When there is no need to read data anymore from a received packet, this function is called instead of lan_get_end function. Although len is the length of data managed by Network Driver Interface and should be used for cancellation, it can be ignored when there is no need to read from the buffer memory of the device. In many devices, its processing is same as lan_get_end function.

Set the length of the packet to send

[Form] **ER lan_set_len(int len)**

len **Number of bytes to send**

[Return Value] **Error Code**

The lan_set_len function of device driver is called from the lan_write_pkt function of the Network Driver Interface to specify the number of bytes of the packet to be transmitted.

Writing the packet to send

[Form] **ER lan_put_pkt(const void *data, int len)**

data **Data to write**

len **Size of data (> 0)**

[Return Value] **Error code**

This function is called from lan_write_pkt function of the Network Driver Interface. Data is written to the buffer memory of the device. This function may be called many times to send one packet (Entire data may not be written at once).

The address of data passed as parameter must be the multiple of 2 (WORD boundry). When lan_put_pkt function is called repeatedly, the value of len must be even, except for the last time.

Depending on the device, it can start the writing operation of a transmitted packet when lan_set_len function is called or it waits for lan_put_end function after a series of continuous lan_put_pkt functions (repeated) and then starts the writing operation. In the case of latter, device driver must have a transmission buffer to store the data passed by lan_put_pkt.

Dummy writing if the packet to be sent is less than 60 bytes

[Form] **ER lan_put_dmy(int len)**

len **Size of dummy data (> 0)**

[Return Value] **Error Code**

The lan_put_dmy function is called before lan_put_end function, if the size of data written with lan_put_pkt function is less than 60 bytes. In the device driver, the data of value “zero” of specified length is written to the buffer memory of the device. This function may not be needed when the device has padding function to automatically compensate the insufficient data to 60 bytes.

End of writing the packet to send

[Form] **ER lan_put_end(void)**

[Return Value] **Error Code**

The lan_put_end function is called after calling lan_put_pkt and lan_put_dmy functions. This starts the transmission from the LAN controller. Even if there is timeout of cancel request in the upper layer, the packet already sent to device driver will be transmitted.

2.4 IP Module

Resources used

IP module consists of 2 tasks, IP Send Task and IP Receive Task. Besides, there is a a mailbox called Send Packet Queue.

In IP Send Task/IP Receive Task, not only IP Module, but ARP, ICMP, IGMP, UDP and TCP Modules are also controlled.

Packet Reception

When IP Receive Task is released by the reception interrupt handler, it receives the packet using the device driver functions. The header of the received packet is analysed here and processing of the packet accourding to ARP, ICMP, IGMP, UDP and TCP protocols, is performed.

Packet Transmission

IP Send Task is released by the packet send to Send Packet Queue. These packets sent to the queue can be packets of any of the protocols ARP, ICMP, IGMP, UDP or TCP.

IP Send Task sets up the IP header of the outgoing packet. Moreover, it sets up the Ethernet header and sends the packet using device driver function. If it cannot send the packet immediately, it waits for wake up from the Transmission interrupt handler.

2.5 ARP Module

Resources used

There is no exclusive task for ARP module. IN ecTCP/IP, IP Send Task/IP Receive Task takes charge of ARP processing. The execution of ARP, a protocol to resolve the MAC address from to IP address of the remote node, cannot be seen from the application.

ARP Table

There is an array in the protocol stack, called ARP Table, to store the MAC address corresponding to an IP address. In ARP Table, the IP address and corresponding MAC address of a destination obtained from the received packet are stored. When the ARP Table gets full, old addresses are removed. In case of timeout error in TCP or UDP reception, the corresponding entry is cleared from ARP Table. Moreover, ARP Table is refreshed after a time interval, set by the configuration of ARP_CACHE_TOUT. The number of members stored in an ARP Table is fixed by the configuration of ARP_TABLE_CNT. Please change this configuration if connection to more number of hosts is reuired.

ARP Request

If destination MAC address for the packet, to be sent by IP Send Task, is not available in ARP Table, IP Send Task sends an ARP packet first. Specifically, an ARP packet is created and sent to Send Packet Queue. Transmission of the original packet is suspended till an ARP response is received.

When IP Receive Task receives the ARP response packet and the address is resolved, IP Send Task transmits the suspended packet.

ARP Response

On the other hand, when IP Receive Task itself receives an ARP packet asking for address, it creates the response packet and sends it to Send Packet Queue. The memory area of the received ARP packet is reused for the memory area of this ARP reply packet.

Timeout of ARP

The memory is stressed when suspended packets with unresolved addresses are accumulated. Therefore, when an address is not resolved after a fixed time interval, the ARP enquiry is sent again and finally, the suspended packet is cancelled. An error is returned to the application task that has sent the cancelled packet. In case of nonblocking call, the application is notified about cancellation by callback function.

2.6 ICMP Module

Resources used

There is no exclusive task for ICMP. IP Send Task/IP Receive Task takes care of ICMP processing. The processing of ICMP protocol, mainly used for network diagnosis, cannot be seen from the application.

Echo process

IP Receive Task that receives the ICMP packet of ECHO used by ping command, create the response packet and sends it to Send Packet Queue. The memory area of the received ICMP packet is reused for the memory area of this ICMP response packet.

icmp_def_cbk

Function Registeration of callback function for ICMP reception

Form **ER icmp_def_cbk(T_ICMP_CB *b, ICMP_CALLBACK callback)**

Return Value **E_OK** **Normal end**
Negative value **Abnormal end**

Description It registers the function to be called when ICMP packets other than ECHO are received.
As ICMP control block is manages by link list, two or more callback functions can be registered. User acquires the memory area for ICMP control block and only passes it as parameter; there is no need for initialization.

Callback

Function Callback for ICMP reception

Form **VP *callback (T_ICMP_CB *b, T_IP *ip, T_ICMP_MSG *icmp, INT len);**

b **Pointer to ICMP control block (Generally not used)**
ip **Pointer to IP packet**
icmp **Pointer to ICMP message**
len **Packet length**

```
typedef struct t_ip {
    struct t_ip *next;          /* Message Pointer for ecRTOS Mailbox */
    T_CTL_HEADER ctl;           /* Header for Internal Control */
    T_ETH_HEADER eth;           /* Ethernet Header */
    T_IP_HEADER ip;             /* IP Header (without option data) */
    B data[1];                  /* Data (Variable Size) */
}T_IP;

typedef struct t_icmp_msg {
```

```
    UB type;            /* ICMP Type */
    UB code;            /* ICMP Code */
    UH cs;              /* Checksum */
    UH id;              /* Identifier */
    UH seq;             /* Sequence Number */
    UB data[IP_HEADER_SZ+8];    /*Option Data (Variable Size)*/
}T_ICMP_MSG;
```

Description This function specifies the ICMP reception callback function. The function name can be anything.

This function is called when ICMP packets other than ECHO, are received.

icmp_snd_dat

Function Transmission of ICMP packet

Form **ER icmp_snd_dat(UW ipaddr, UW dstaddr, T_ICMP_HEADER *icmp, VP data, INT len);**

ipaddr **Local IP Address**

dstaddr **Remote IP Address**

icmp **Pointer to the ICMP header to be sent**

data **Pointer to the ICMP data to be sent**

len **Data Length**

```
typedef struct t_icmp_header {
    UB type;            /* ICMP Type */
    UB code;            /* ICMP Code */
    UH cs;              /* Checksum */
    UH id;              /* Identifier */
    UH seq;             /* Sequence Number */
}T_ICMP_HEADER;
```

Return Value **E_OK** **Normal end**

Negative value **Abnormal end**

Description Any ICMP packet can be transmitted and it can also be called from the above mentioned callback function. Implementation of ping_command is an example for this.

2.7 UDP Module

Resources used

There is no exclusive task for UDP. In ecTCP/IP, IP Send Task/IP Receive Task takes care of the UDP processing. There is a separate mailbox called UDP Reception Queue for each UDP Communication End Point.

Transmission of UDP packet

When application requests for transmission of a UDP packet using this service call, UDP header is added to it and sent to Send Packet Queue. If IP Send Task finished transmission of this UDP packet, the application task that was waiting for completion of transmission, is released. In case of nonblocking call, the completion of transmission is notified to the application by callback.

Reception of UDP packet

In UDP reception, the application task specifies the memory area for reception of UDP packet before using the service call.

IP Receive Task that received the UDP packet, analyses the UDP header of the packet. Then, if there is a reception request from the corresponding UDP Communication End Point, the packet is copied to the specified memory area. When the UDP packet is received in this service call, it releases the task that was waiting for reception.

In case of nonblocking call, the completion of reception is notified to the application by the callback function.

When there is no request for UDP reception, the callback function is called to notify the reception of UDP packet.

Important!
UDP, unlike TCP, does not hava and internal buffer to store the data. Therefore, if protocol stack receives a UDP packet and there is no application waiting in udp_rcv_dat at that time, the packet is ignored. Please a use callback function, when UDP packets are received continuously. If a callback function has reception, it is called from the protocol stack. This process is similar to interrupt handler.

2.8 TCP Module

Resources used

There is no exclusive task for TCP. In ecTCP/IP, IP Send Task/IP Receive Task takes care of TCP processing.

Relation with IP Module

In IP Receive Task, the IP address and Port number of destination/transmitter of received TCP packet are distinguished and the TCP packet is processed by the TCP Communication End Point, if a connected (including connecting) TCP Communication End Point exists.

If a TCP packet is connection request (SYN) and there is a corresponding Reception Point, the connection of TCP Communication End Point is established. All the packets, for nonexisting connection and SYN packets, for nonavailable Reception Points, are ignored.

In IP Send Task, IP header is set in the TCP packet sent from TCP module and the packet is transmitted.

IP Receive Task copies the TCP data to Receive buffer managed by TCP module. Similarly, IP Send Task reads the TCP data from the Send buffer.

Keep Alive

After TCP connection establishment, if there is no communication for a fixed time interval, a Keep-alive packet is transmitted. Its function is to check whether the remote host is operating normally. A Keep-alive Packet is the packet with only ACK flag set to ON.

If the remote host is working normally, it replies the response for this packet. The connection is cut if there is no response. In the case of reception function, the error is returned at this time by 0 or E_CLS. The time interval for sending Keep-alive can be configured using TCP_KTIME_INI.

Stream based communication

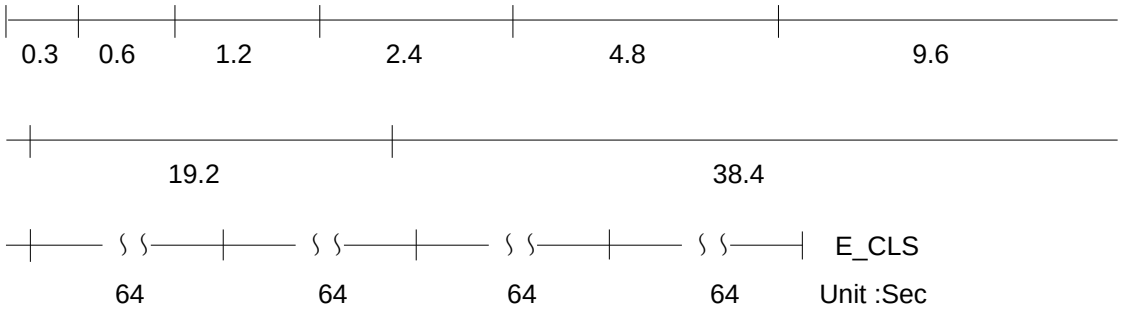
As opposed to UDP and IP sending packet of one block called datagram, TCP has a system to transmit and receive data sequence with no packet boundries. This is called Stream based communication system. For example, when 1000 bytes of data is transmitted, the received data is same as the sent data when reception side receives entire 1000 bytes at once, or it is received as devided into 500 bytes units. When it is required to receive data as one block, it is done by specifying the length at the head of the packet, dividing the packet and attaching a character at the end of the packet.

Retransmission of messages

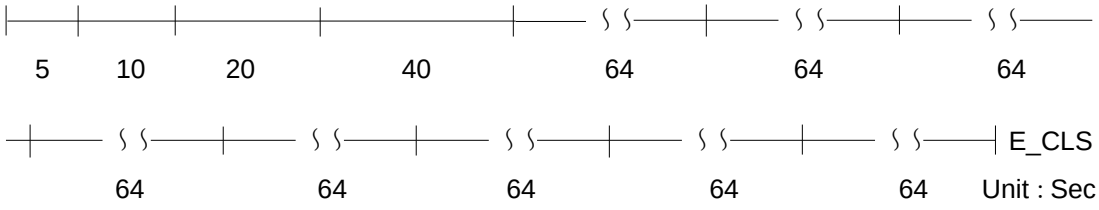
In IP network where reliability is not guaranteed, a packet may be lost during transmission. To

recover from such error in TCP, where ACK packet is not received from the remote host in a fixed time interval and it is timed out, the same packet is transmitted. This timeout interval is measured by the response time of the opponent from the time of packet transmission (Round trip time), the speed of entire route changes dynamically according to the path taken. In ecTCP/IP, the timeour interval can be configured using TCP_RTO_UBOUND for maximum value and TCP_RTO_LBOUND for minimum value. When ACK is not received successively, the message is retransmitted again but its interval is doubled from last time. For example in the case of data and FIN transmission, if there is no ACK response from the other end, it will retransmit the message at following intervals.

(Ethernet example) ※



(PPP example using 9600bps) *



* This is a typical example evaluated in the company. Since the timeout interval is based on theround-trip time, the time changes with the speed of the route used. When **API** with timeout is used and processing is not continued within specified time, the retransmission is continued, although **E_TMOUT** is returned to the calling function. If **tcp_cls_cep** is called, with short timeout (about **10MSEC**) **RST** is sent a nd retransmission of messages is stopped.

Chapter 3 Configuration

Definition

Before calling #include for the configuration header ecnetc.h, the configuration can be performed with the constants described as follows:

() shows the default value which is used when there is no specification. However, user must specify values for ?.

#define TCP_REPID_MAX	**(4)**	**Upper limit of TCP Reception Point**
#define TCP_CEPID_MAX	**(4)**	**Upper limit of TCP Communication End Point**
#define UDP_CEPID_MAX	**(4)**	**Upper limit of UDP Communication End Point**
#define TCP_TSKID_TOP	**?**	**Top ID of the task used in TCP/IP**
#define TCP_SEMID_TOP	**?**	**Top ID of semaphore used in TCP/IP**
#define TCP_MBXID_TOP	**?**	**Top ID of the mailbox used by TCP/IP**
#define TCP_MPFID_TOP	**?**	**Top ID of the fixed-length memory pool used by TCP/IP**
#define ARP_TABLE_CNT	**(8)**	**Number of entried in ARP Table**
#define PRI_IP_SND_TSK	**(4)**	**IP Send Task priority**
#define PRI_IP_RCV_TSK	**(4)**	**IP Receive Task priority**

Important !

The priority of IP Send and Receive Tasks must be same, please set this priority higher than the priority of the application task that uses the protocol stack. When Send and Receive Tasks IP are lower than the application task, it may cause unexpected operations.

#define SSZ_IP_SND_TSK	**(1024)**	**Stack size of IP Send Task**
#define SSZ_IP_RCV_TSK	**(1024)**	**Stack size of IP Receive Task**
#define ETH_QCNT	**(16)**	**Maximum queue count for Ethernet packets*1**
#define UDP_QCNT	**(2)**	**Maximum queue count for UDP packets**

#define ARP_CACHE_TOUT	**(1200)**	**Refresh time interval of ARP Table (20 min)**
#define ARP_RET_INTVAL	**(2)**	**Resend interval of ARP (2 sec)**
#define IP_DEF_TTL	**(32)**	**TTL(Time To Live)**
#define TCP_SYN_RETRY	**(3)**	**Maximum number of retries of TCP SYN transmission**
#define TCP_DATA_RETRY	**(12)**	**Maximum number of retries of TCP data transmission**
#define TCP_RTO_INI	**(3000)**	**Initial value of TCP resend timeout (3 Sec)**
#define TCP_RTO_UBOUND	**(64000)**	**Upper limit of TCP resend timeout (64 Sec)**
#define TCP_RTO_LBOUND	**(300)**	**Lower liit of TCP resend timeout (300 mSec)**
#define TCP_KTIME_INI	**(7200)**	**Time untill Keep-alive packet is transmitted (2 hours)** **In case of 0 or less, keep-alive packet is not transmitted※2**
#define TCP_KTIME_PRO	**(600)**	**Keep-alive timeout (10 min)**
#define TCP_KTIME_SUC	**(75)**	**Transmission interval of keep-alive packets (75 Sec)**

#include "ecnetc.h"

*1ETH_QCNT is used for IP fragment processing as well. With default value, data upto 20Kbytes can be processed. If UDP or ICMP data exceeding this limit is expected to be received at a time, please increase the value.

*2Value of TCP_KTIME_INI should not be set less than TCP_RTO_UBOUND

Automatic assignment of ID

ecTCP/IP supports automatic assignment of each resource ID.

When the top ID of each resource is set as 0, all internal IDs are automatically assigned by ecTCP/IP.

#define TCP_TSKID_TOP	**0**	**Top ID of the task used by TCP/IP**
#define TCP_SEMID_TOP	**0**	**Top ID of the semaphore used by TCP/IP**
#define TCP_MBXID_TOP	**0**	**Top ID of the mailbox used by TCP/IP**
#define TCP_MPFID_TOP	**0**	**Top ID of the fixed-length mempry pool used by TCP/IP**

Automatic ID assignment of Reception Point

When xxx_cre_xxx system call creates an object, unoccupied ID number is returned for acquisition.

xxx_vcre_xxx system call is an extended function, original to ecTCP/IP.

Setup of local IP Address and MAC address

Local IP address and MAC address are defined using following variable:

```
UB default_ipaddr[]   =  { 192, 168, 1, 99 };
UB ethernet_addr[]    =  { 0x00, 0x00, 0x12, 0x34, 0x56, 0x78 };
```

It is necessary to setup these variables before any service call is called.

Setup of default gateway and subnet mask

The following variable define default gateway and subnet mask:

```
UB default_gateway[]  =  { 0, 0, 0, 0 };
UB subnet_mask[]      =  { 255,255,255,0 };
```

It is necessary to setup these variables before any service call is called.

When gateway is not used, please set it as all 0s.

Chapter 4 Common Definition

4.1 Byte-order conversion

The form of the data of TCP/IP header passing on the network (network byte order), is big endian. In case the form of data of processor (host byte order) is little endian, it needs conversion. Since this conversion is performed inside the protocol stack in ecTCP/IP, there is no need to take care for the byte order in the application.

When byte order conversion of TCP data or UDP data is done in the application, the following utility functions/macros can be used.

htonl	**Host→Network Byte Order Conversion (long)**
htons	**Host→Network Byte Order Conversion (short)**
ntohl	**Network→Host Byte Order Conversion (long)**
ntohs	**Network→Host Byte Order Conversion (short)**

4.2 Extraction of Error Code

As a general rule, the service call of ecTCP/IP, return error code as return value. In the lower byte of the error code, error codes common to ITRON specification (Main Error Code) are contained. In the higher byte of the error code, TCP/IP specific error codes (Sub Error Code) are included.

All error codes in the return value of service call, Main Error Code and Sub Error Code, are negative values. There are 2 MACROs available to take out only Main Error Code or only Sub Error Code from the error code retunred by service call:

mainercd **Extraction of Main Error Code**
subercd **Extraction of Sub Error Code**

*In ecTCP/IP, Sub Error Code is not defined.

4.3 Structure

```
typedef struct {
    UW ipaddr;              IP Address
    UH portno;              Port Number
} T_IPV4EP;

typedef struct {
    ATR repatr;             TCP Reception Point attributes (Unused, 0)
    T_IPV4EP myaddr;        Local IP Address and Port Number
} T_TCP_CREP;

typedef struct {
    ATR cepatr;             TCP Communication End Point attribute (Unused, 0)
    VP sbuf;                Address of the top of Send Buffer
    INT sbufsz;             Size of Send Buffer
    VP rbuf;                Address of the top of Receive Buffer
    INT rbufsz;             Size of Receive Buffer
    FP callback;            Address of callback routine
} T_TCP_CCEP;

typedef struct t_udp_ccep {
    ATR cepatr;             UDP Communication End Point attribute (Unused, 0)
    T_IPV4EP myaddr;Local IP Address and Port Number
    FP callback;            Address of callback routine
} T_UDP_CCEP;
```

4.4 Main Error Codes

E_OK	0	0		Normal End
E_SYS	0xf..ffb	-5		System Error
E_NOSPT	0xf..ff7	-9	(-17)	Unsupported function
E_PAR	0xf..fef	-17	(-33)	Parameter Error
E_ID	0xf..fee	-18	(-35)	Inaccurate ID Number
E_NOMEM	0xf..fdf	-33	(-10)	Memory unavailable
E_NOEXS	0xf..fd6	-42	(-52)	Uncreated object
E_OBJ	0xf..fd7	-41	(-63)	Object state error
E_QOVR	0xf..fd5	-43	(-73)	Queuing overflow
E_RLWAI	0xf..fcf	-49	(-86)	Process cencelled, forced release of WAIT state
E_TMOUT	0xf..fce	-50	(-85)	Polling failure or Timeout
E_DLT	0xf..fcd	-51	(-81)	Waiting object deleted
E_WBLK	0xf..fad	-83		Nonblocking call received
E_CLS	0xf..fa9	-87		Connection is closed
E_BOVR	0xf..fa7	-89		Buffer overflow

The value within () are µITRON3.0 specified.

Chapter 5 Utility/Macro

5.1 Utility/Macro

htonl

Function Host→Network byte order conversion (long)

Form **UW htonl(UW hl);**

Hl **Data in Host byte order**

Return Value Data converted to Network byte order

Description Long WORD data (32 bits) in Host byte order is converted to Netwrok byte order.

htons

Function Host→Network byte order conversion (short)

Form **UH htons(UH hs);**

Hs **Data in Host byte order**

Return Value Data converted to Network byte order

Description Short WORD data (16 bits) in Host byte order is converted to Network byte order.

ntohl

Function Network→Host byte order conversion (long)

Form **UW ntohl(UW nl);**

Nl **Data in Network byte order**

Return Value Data converted to Host byte order

Description Long WORD data (32 bits) in Network byte order is converted to Host byte order.

ntohs

Function Network→Host byte order conversion (short)

Form **UH ntohs(UH ns);**

Ns **Data in Netowrk byte order**

Return Value Data converted into Host byte order

Description Short WORD data (16 bits) in Netowrk byte order is converted to Host byte order.

5.2 Utility functions

byte4_to_long

Function	4 bytes array of IP address →long conversion
Form	**UW byte4_to_long(const UB *b);**
	const UB *b 4 byte array to be coverted
Return Value	The converted value
Description	4 byte array of IP address is converted into long type IP Address

long_to_byte4

Function	long→4byte array of IP Address conversion
Form	**void long_to_byte4(UB *b, UW d);**
	UB *b the pointer to 4 byte array
	UW d long value to be converted
Description	long type IP Addres is converted into 4 byte array of IP Address.

ascii_to_ipaddr

Function	String of IP Address →long type IP Address conversion
Form	**UW ascii_to_ipaddr(B *s);**
	B *s String of IP Address to be converted
	(Example)"192.168.100.99 "
Return Value	The converted value
Description	IP string is converted into long type IP Address.

ipaddr_to_ascii

Function	long type IP Address →IP Address string
Form	**INT ipaddr_to_ascii(B *s, UW ipaddr);**
	B *s Pointer to the string for converted IP address
	UW ipaddr IP Address to be changes
Return Value	Size of the converted string
Description	long type IP Address is converted to IP Address string.

Chapter 6 TCP Service Call

List of TCP service call

tcp_cre_rep	Creation of TCP Reception Point
tcp_vcre_rep	Creation of TCP Reception Point (Automatic ID allocation)
tcp_del_rep	Deletion of TCP Reception Point
tcp_cre_cep	Creation of TCP Communication End Point
tcp_vcre_cep	Creation of TCP Communication End Point (Automatic ID allocation)
tcp_del_cep	Deletion of TCP Communication End Point
tcp_acp_cep	Wait for connection request (Passive open)
tcp_con_cep	Connection request (Active open)
tcp_sht_cep	End of data transmission
tcp_cls_cep	Closing of TCP Communication End Point
tcp_snd_dat	Transmit data
tcp_rcv_dat	Receive data
tcp_get_buf	Acquire Send buffer (Reduced copy API)
tcp_snd_buf	Send data from the buffer (Reduced copy API)
tcp_rcv_buf	Acquire buffer containing received data (Reduced copy API)
tcp_rel_buf	Release receiving buffer (Reduced copy API)
tcp_snd_oob	Send urgent data
tcp_rcv_oob	Receive urgent data
tcp_can_cep	Cancel TCP process
tcp_set_opt	Setup TCP Communication End Point options
tcp_get_opt	Refer to TCP Communication End Point options

tcp_cre_rep

Function Create TCP Reception Point

Form

```
ER tcp_cre_rep(ID repid, T_TCP_CREP *pk_crep);
repid      TCP Reception Point ID
pk_crep    TCP Reception Point creation information

typedef struct {
        ATR repatr;             TCP Reception Point attribute (Unused, 0)
        T_IPV4EP myaddr;        Local IP Address and Port Number
} T_TCP_CREP;

typedef struct {
        UW ipaddr;              IP Address
        UH portno;              Port Number
} T_IPV4EP;
```

Return Value

E_OK	**Normal End**
E_ID	**Incorrect ID Number**
E_OBJ	**TCP Reception Point already created or Port Number already used**
E_PAR	**Invalid IP Address or Port Number specified**

Description This creates the TCP Reception Point specified by repid.

The created TCP Reception Point waits only for the connection request with IP Address specified as pk_crep->myaddr.ipaddr and Port Number specified as pk_crep->myaddr.portnumber.

Supplement It does not support the case when local IP Address is specified as IPV4_ADDRANY(=0) and it has to wait for the connection request on all local IP Addresses.

tcp_vcre_rep

Function Creation of TCP Reception Point (Automatic ID allocation)

Form **ER tcp_vcre_rep(T_TCP_CREP *pk_crep);**

pk_crep **yTCP Reception Point creation information**

Return Value

Positive value	**The assigned Reception Point ID**
E_ID	**Reception Point ID unavailable**
E_OBJ	**Port Number already existing**
E_PAR	**Invalid IP Address or Port Number specified**

Description It searches for an unused Repection Point ID and allocates it. When Reception Point ID cannot be assigned, it returns E_ID error. Except this, it is same as tcp_cre_cep.

Supplement This systems call is original to ecTCP/IP.

tcp_del_rep

Function Deletion of TCP Reception Point

Form **ER tcp_del_rep(ID repid);**

	repid	**TCP Reception Point ID**

Return Value

E_OK	**Normal End**
E_ID	**Incorrect ID number**
E_NOEXS	**Non-existing TCP Reception Point**

Description It deletes TCP Reception Point specified by repid. Since the connection waiting process, queued by this TCP Reception Point, is cancelled, an E_DLT error is retuned to the task that called the tcp_acp_cep service call. When tcp_acp_cep is called with nonblocking specification, E_DLT error is notified to the application by the callback.

tcp_cre_cep

Function Creationg TCP Communication End Point

Form **ER tcp_cre_cep(ID cepid, T_TCP_CCEP *pk_ccep);**

cepid **TCP Communication End Point ID**

pk_ccep **Pointer to structure for creation information of TCP Communication End Point**

```
typedef struct {
    ATR cepatr;     TCP Communication End Point attribute (Unused, 0)
    VP sbuf;        Top address of Send buffer
    INT sbufsz;     Size of Send buffer
    VP rbuf;        Top address of Receive buffer
    INT rbufsz;     Size of Receive buffer
    FP callback;    The address of callback routine
} T_TCP_CCEP;
```

Return Value

E_OK	**Normal End**
E_ID	**Incorrect ID number**
E_OBJ	**TCP Communication End Point already created**
E_PAR	**Specification of buffers (Send/Receive) and size are not correct**

Description This creates the TCP Communication End Point specified by cepid. Here TCP Communication End Point is generated only, it cannot function yet. It can function only after it is open by service call tcp_acp_cep for passive opening or tcp_con_cep for active opening.

The specified buffer areas are managed inside the protocol stack. Service call of Reduced copy API tcp_get_buf and tcp_rcv_buf retunrs pointer to either of these areas. When this pointer is not used, direct access to buffer memory should be avoided. Moroever, the buffer area should use the memory that is independent of Communication End Point.

The queuing limit of requests to send and receive packets is decided by the configuration. The requested and received packets exceeding this limit are ignored.

Supplement It does not support the case to acquire the buffer inside the protocol stack when NADR is specified as top address of the buffer memory.

It is necessary to specify the size of Send/Receive buffer as more than 2 bytes. Please use the largest possible buffer to perform more efficient communication.

Communication efficiency will become worse if size of Send/Receive buffer is too small. When large data is transmitted and received continuously, it is effective to setup 2048 and 8192 as minimum limit of buffer size to perform optiomal communication.
When the total data size of a continous communication, in one connection, exceeds 4Gbyte, please specify the size of Send/Receive buffer in the power of 2.

tcp_vcre_cep

Function Creation of TCP Communication End Point (Automatic ID allocation)

Form **ER tcp_vcre_cep(T_TCP_CCEP *pk_ccep);**

pk_ccep **Pointer to structure for creation information of TCP Communication End Point**

Return Value

Positive value **The assigned ID of Communication End Point**

E_ID **TCP Communication End Point is unavailable**

E_PAR **Incorrect specification of buffer (Send/Receive) and size**

Description This searches for an unused TCP Communication End Point ID and allocates it. When TCP Communication End Point cannot be assigned, it returns E_ID as error. Except this, it is same as tcp_cre_cep.

Supplement This is ecTCP/IP original system call.

tcp_del_cep

Function Deletion of TCP Communication End Point

Form **ER tcp_del_cep(ID cepid);**

	cepid	**TCP Communication End Point ID**

Return Value

E_OK	**Normal End**
E_ID	**Incorrect ID Number**
E_NOEXS	**Non-existing TCP Communication End Point**
E_OBJ	**TCP Communication End Point is being used**

Description This deletes the TCP Communication End Point specified by cepid. When TCP Communication End Point is being used, i.e. open-wait~closing, it cannot be deleted.

tcp_acp_cep

Function Wait for connection request (Passive open)

Form **ER tcp_acp_cep(ID cepid, ID repid, T_IPV4EP *p_dstaddr, TMO tmout);**

cepid **TCP Communication End Point ID**

repid **TCP Reception Point**

dstaddr **Pointer to the structure for remote IP Address/Port number**

tmout **Timeout specification**

```
typedef struct {
    UW ipaddr;      IP Address
    UH portno;      Port Number
} T_IPV4EP;
```

Return Value

E_OK	**Normal End**
E_ID	**Incorrect ID number**
E_NOEXS	**Non-existiong TCP Communication End Point**
E_OBJ	**TCP Communication End Point is already in use**
E_DLT	**TCP Reception Point was deleted while waiting for connection request**
E_WBLK	**Nonblocking call accepted**
E_TMOUT	**Polling falied or timed out**
E_RLWAI	**Prcoess cancelled, Wait state released forcibly**

Description By this service call, TCP Communication End Point specified by cepid, goes into passice open Wait state. Thus, TCP Reception Point specified by repid, waits for to receive the connection request. If SYN is received, TCP Communication End Point sends ACK to complete the connection (connection is established) and goes into the Connection state.

Remote IP Address and Port Number are retunred in *dstaddr.

When this service call is issued with no timeout (tmout = TMO_FEVR), the issuing task goes into Wait state till the connection is completed.

When the service call is issed with a timeout value (tmout = 1～0x7fffffff), E_TMOUT error is returned if the specifed time is passed and there is no connection request or the

connection is not completed.

When the service call is issued as nonblocking (tmout = TMO_NBLK), completion of the connection is notified using the callback function.

Two or more tcp_acp_cep service calls can be issued simultaneously for the same TCP Reception Point. In that case, the connection is accepted by the TCP Communication End Point, issued first.

When processing of this service call is cancelled by a timeout or tcp_can_cep, TCP Communication End Point returns to Unused state.

Supplement When this service call is issued as nonblocking (tmout = TMO_NBLK), writing to the specified pointer of the structure for remote IP Address/Port Number (dstaddr) is done after connection, hence please do not use stack space for this memory.

tcp_con_cep

Function Connection request (Active open)

Form **ER tcp_con_cep(ID cepid, T_IPV4EP *p_myaddr, T_IPV4EP *p_dstaddr, TMO tmout);**

cepid **TCP Communication End Point ID**

myaddr **Pointer of the structure for local IP Address/Port Number**

dstaddr **Pointer of the structure for remote IP Address/Port Number**

tmout **Timeout specification**

```
typedef struct {
    UW ipaddr;      IP Address
    UH portno;      Port Number
} T_IPV4EP;
```

Return Value

E_OK	**Normal end**
E_ID	**Incorrect ID number**
E_NOEXS	**Non-existing TCP Communication End Point**
E_OBJ	**TCP Communication End Point or Port Number is already in use**
E_WBLK	**Nonblocking call accepted**
E_TMOUT	**Polling failed or timed out**
E_RLWAI	**Process cancelled or Wait state release forcibly**
E_CLS	**The connection request is refused**

Description In this service call, TCP Communication End Point specified by cepid goes into active open Wait state. That is, SYN is sent from TCP Communication End Point to the remote IP Address and Port Number specified by *dstaddr and connection is requested. When ACK is received, the connection is completed (established) and Communication End Point will go into Connection state.

When this service call is issued with no-timeout specification (tmout = TMO_FEVR), the issuing task will be in Wait state till the connection is completed.

When this service call is issued with a timeout value (tmout = 1～0x7fffffff), E_TMOUT

error is returned when the specified time passes and the connection is not completed.

Since polling (tmout = TMO_POL) is meaningless, E_TMOUT error is retunred.

When this service call is issued as nonblocking (tmout = TMO_NBLK), the completion of the connection is notified by the callback function.

When processing of tcp_con_cep is cancelled by timeout or tcp_can_cep or if connection request is rejected, the TCP Communication End Point returns to Unused state.

When local IP Address is specified as IPV4_ADDRANY(= 0), protocol assigns the value of default_ipaddr for it. When Port Number is assigned as TCP_PORTANY (=0), protocol stack assigns an arbitrary value for it.

Supplement It does not support the function to determine the IP Address and Port Number inside the protocol stack when myaddr is specified as NARD(= -1).

When this service call is issued with no-timeout specification (tmout = TMO_FEVR) and there is no response from the remote IP Address, the process does not return from this service call.

When E_CLS is returned, it signifies rejection of connection request (RST reception) from the remote end. In this case, please check whether the port of the desctination is prepared or whether the Port Number is correct.

tcp_sht_cep

Function The end of data transmission

Form **ER tcp_sht_cep(ID cepid);**

	cepid	**TCP Comunication End Point ID**

Return Value

E_OK	**Normal End**
E_ID	**Incorrect ID number**
E_NOEXS	**Non-existing TCP Communication End Point**
E_OBJ	**TCP Communication End Point is not connected**

Description To requests the end of data transmission from the TCP Communication End Point specified by cepid. TCP Communication End Point immediately goes into send Termination state and sends FIN when it finishes transmitting the data in the send buffer. This service call only changes the state of the TCP Commnuication End Point but the issuing task does not go into Wait state.

This service call is used to inform the remote host about the end of data transmission.

Remote end that receives FIN, knows that data will not be sent anymore and terminates the connection (closing) after sending the last response.

tcp_cls_cep

Function Closing of Communication End Point

Form **ER tcp_cls_cep(ID cepid, TMO tmout);**

	cepid	**TCP Communication End Point ID**
	tmout	**Timeout specification**

Return Value

E_OK	**Normal End**
E_ID	**Incorrect ID number**
E_NOEXS	**Non-existing TCP Communication End Point**
E_OBJ	**TCP Communication End Point is not connected**
E_WBLK	**Nonblocking call received**
E_TMOUT	**Polling failure or timed out**
E_RLWAI	**Process cancelld or Wait state released forcibly**

Description The connection of TCP Communication End Point specified by cepid is closed.

When TCP Communication End Point has not terminated the transmission, it waits to send the FIN till the data in the send buffer is sent. Then, it waits for the reception of ACK for the sent FIN. On the other hand, if the remote end has not finished the transmission, it waits for the reception of FIN while dumping the received data and sends ACK when FIN is received.

Process of this service call is completed as the above disconnection process and the TCP Communication End Point goes into Unused state.

When this service call is issued with no-timeout specification (tmout = TMO_FEVR), the issuing task is in Wait state till the disconnection is complete. Since TCP Communication End Point is in Unused state after returning from this service call, it can be reused immediately.

When this service call is issued with a timeout value (tmout = 1～0x7fffffff), E_TMOUT is returned when the specified time passes and the disconnection is not complete. In this case and when the processing is cancelled by tcp_can_cep, TCP Communication

End Point sends RST and forcibly disconnects the connection.

Since polling (tmout = TMO_POL) is meaningless, E_TMOUT error is returned.

When this service call is issued as nonblocking (tmout = TMO_NBLK), the completion of disconnection is notified using the callback.

Supplement Since the callback function may not be called when TMO_NBLK is specified and the remote end carshes, as far as possible, please specify timeout interval. Even if E_TMOUT error is returned, it is safely disconnected after sending RST.

tcp_snd_dat

Function Transmission of data

Form **ER tcp_snd_dat(ID cepid, VP data, INT len, TMO tmout);**

cepid	**TCP Communication End Point ID**
data	**Pointer to the data to be sent**
len	**Length of data to be sent**
tmout	**Timeout specification**

Return Value

Positive Value	**Normal End (Length of data copied into send buffer)**
E_ID	**Incorrect ID number**
E_NOEXS	**Non-existing TCP Communication End Point**
E_OBJ	**TCP Communication End Point is not connected or terminated or transmission is pending**
E_WBLK	**Nonblocking call received**
E_TMOUT	**Polling failure or timed out**
E_RLWAI	**Process cancelled or Wait state released forcibly Or, no ARP response from the specified IP Address**
E_CLS	**TCP connection closed**
E_PAR	**Length in len is invalid**

Description This sends the data from the TCP Communication End Point specified by cepid. This service call returns after copying the data (pointed by “date”) of length “len” only to the send buffer. If the space in the send buffer is less than the length of data to be sent, it copies data into send buffer till it gets full and returns the size of the copied data. When there is no space in the send buffer, the issuing task is in Wait state, till some space is created.

When this service call is issued with no-timeout (tmout = TMO_FEVR) specification, the issuing task is in Wait state till the data is completely copied to the send buffer.

When this service call is issued with timeout value (tmout = 1～0x7fffffff), E_TMOUT error is returned when the specified time passes and there is no space in the send buffer.

When this service call is issued with polling specification (tmout = TMO_POL) and there is no space available in the send buffer, an E_TMOUT error is returned immediately.

When this service call is issued as nonblocking (tmout = TMO_NBLK) and there is no space in the send buffer, the service call is returned immediately and the completion of data copy to the send buffer is notified by the callback.

Send process caanot be queued. If same TCP Communication End Point receives tcp_snd_dat or tcp_get_buf twice, E_OBJ error is returned.

Supplement The legths (len) of transmission data, specified as parameter and the return value are not necessarily same. When the space in send buffer is less than the length (len) of transmission data, the data equal to the empty buffer size is copied.

tcp_rcv_dat

Function Reception of data

Form **ER tcp_rcv_dat(ID cepid, VP data, INT len, TMO tmout);**

cepid	**TCP Communication End Point ID**
data	**Pointer to the area where received data is copied**
len	**Length of data to receive**
tmout	**Timeout specification**

Return Value

Positive Value	**Normal end (Length of received data)**
0	**End of data (Connection terminated normally)**
E_ID	**Incorrect ID number**
E_NOEXS	**Non-existing TCP Communication End Point**
E_OBJ	**TCP Communication End Point is not connected or reception is pending**
E_WBLK	**Nonblocking call received**
E_TMOUT	**Polling failure or timed out**
E_RLWAI	**Process cancelled or Wait state released foricbly**
E_CLS	**TCP connection is closed or receive buffer is empty**
E_PAR	**Length (len) is invalid**

Description It receives data from TCP Communication End Point specified by cepid. This service call returns after the data held in Receive buffer is copied to the area pointed by “data”. When data held in the Receive buffer is shorter than the length “len” of the data to be received, data is extracted until the Receive buffer is empty and the length of extracted data is returned as return value.

When the Receive buffer is empty, the issuing task of this service call remains in Wait state until some data is received.

When this service call is issued with no-timeout specification (tmout = TMO_FEVR), the issuing task will be in Wait state until the data copy is completed.

When this service call is issued with a timeout value (tmout = 1～0x7fffffff), E_TMOUT error is returned if the specified time passes and no data is received.

When this service call is issued in polling (tmout = TMO_POL) mode, E_TMOUT error is retuned immediately if there is no data in the Receive buffer.

When this service call is issued as nonblocking (tmout = TMO_NBLK) and there is no data in the Receive buffer, the service call is immediately returned and completion of data reception is notified by the callback.

Reception process cannot be queued. If tcp_rcv_dat or tcp_rcv_buf are called twice for the same TCP Communication End Point, E_OBJ error is returned.

When the connection is closed normally from the remote host and there is no data in the Receive buffer, this service call returns 0.

tcp_get_buf

Function Acquisition of Send buffer (Reduced Copy API)

Form **ER tcp_get_buf(ID cepid, VP *p_buf, TMO tmout);**

cepid	**TCP Communication End Point ID**
buf	**Pointer to the top address of the empty area**
tmout	**Timeout specification**

Return Value

Positive Value	**Normal end (Length of empty area)**
E_ID	**Incorrect ID number**
E_NOEXS	**Non-existing TCP Communication End Point**
E_OBJ	**TCP Communication End Point is not connected or sending terminated or transmission is pending**
E_WBLK	**Nonblocking call received**
E_TMOUT	**Polling failure or timed out**
E_RLWAI	**Process cancelled or Wait state released forcibly**
E_CLS	**TCP connection is closed**

Description From the Send buffer of TCP Communication End Point specified by cepid, the top address of the empty area where data to be transmitted can be copied and the length of area are returned. When there is no space in the Send buffer, the calling task of this service call will be in Wait state until some space is available.

When this service call is issued with no-timeout specification (tmout =TMO_FEVR), the issuing task will be in Wait state till an empty area is acquired.

When this service call is issued with a timeout value (tmout = 1～0x7fffffff), E_TMOUT error is returned when the specified time passes and the empty area is not available.

When this service call is issued in polling mode (tmout = TMO_POL) and there is no empty area available, E_TMOUT error is returned immediately.

When this service call is issued as nonblocking (tmout = TMO_NBLK) and there is no empty area available, the service call returns immediately and the completion of acquisition of empty area is notified by the callback.

By issuing this service call, the internal state of the protocol stack does not change. Therefore, the same area is returned if tcp_get_buf is called repeatedly. On the other hand, calling tcp_snd_dat and tcp_snd_buf changes the internal state of the protocol stack. By these calls, the information returned by earlier tcp_get_buf will become invalid.

The send process cannot be queued. When tcp_snd_dat or tcp_get_buf is issued twice for the same TCP Communication End Point returns E_OBJ error.

tcp_snd_buf

Function Transmission of data in the buffer (Reduced Copy API)

Form **ER tcp_snd_buf(ID cepid, INT len);**

	cepid	**TCP Communication End Point ID**
	len	**Length of data**

Return Value

E_OK	**Normal end**
E_ID	**Incorrect ID number**
E_NOEXS	**Non-existing TCP Communication End Point**
E_OBJ	**TCP Communication End Point is not connected or transmission terminated or length "len" is too long**
E_CLS	**TCP connection is closed**
E_PAR	**Length "len"is iunvalid**

Description It sends the data of length "len", written in the buffer acquired by tcp_get_buf, from the TCP Communication End Point specified by cepid. As tcp_snd_buf only requires transmission, the issuing task of this service call will not be in Wait state.

tcp_rcv_buf

Function Acquisition of buffer containing received data (Reduced Copy API)

Form **ER tcp_rcv_buf(ID cepid, VP *p_buf,TMO tmout);**

cepid **TCP Communication End Point ID**

buf **Pointer to the top address of the receive buffer**

tmout **Timeout specification**

Return Value

Positive Value	**Normal end (Length of received data)**
0	**End of data (Normal termination of connection)**
E_ID	**Incorrect ID number**
E_NOEXS	**Non-existing TCP Communication End Point**
E_OBJ	**TCP Communication End Point not connected or reception is pending**
E_TMOUT	**Polling failure or timed out**
E_RLWAI	**Process cancelled or Wait state released forcibly**
E_CLS	**TCP connection is terminated or Receive buffer is empty**
E_WBLK	**Nonblocking call received**

Descriptiopn It sets the top address of the area containing the data received by TCP Communication End Point, specified by cepid, to *buf and returns the length of the data as return value. When the Receive buffer is empty, the issuing task of this service call will be in Wait state until some data is received.

When this service call is issued with no-timeout specification (tmout = TMO_FEVR), the issuing task will be in Wait state until the data is received.

When this service call is issued with a timeout value (tmout = 1 ～0x7fffffff), E_TMOUT is returned when the specified time passes and no data is received.

When this service call is issued in polling mode (tmout = TMO_POL) and there is no received data, E_TMOUT error is returned immediately.

When this service call is issued as nonblocking (tmout = TMO_NBLK) and there is no received data, the service call is returned immediately and the data reception is notified by the callback.

The internal state of the protocol stack does not change when this servie call is issued. Therefore, same area is returned when tcp_rcv_buf is called repeatedly. On the other hand, calling tcp_rcv_dat and tcp_rel_buf change the internal state of the protocol stack. If these service calls are issued, the information returned by previous tcp_rcv_buf becomes invalid.

If the connection is closed normally by the remote host and there is no data left in the Receive buffer, this service call returns 0.

Even when the connection is terminated abnormally but there is data in the Receive buffer, the top address and length of the data can be extracted.

Reception process cannot be queued. When tcp_rcv_dat or tcp_rcv_buf are issued twice for the same TCP Communication End Point, E_OBJ error is returned.

tcp_rel_buf

Function Release of Receive buffer (Reduced Copy API)

Form **ER tcp_rel_buf(ID cepid, INT len);**

	cepid	**TCP Communication End Point ID**
	len	**Length of data**

Return Value

E_OK	**Normal end**
E_ID	**Incorrect ID number**
E_NOEXS	**Non-existing TCP Communication End Point**
E_OBJ	**Specified TCP Communication End Point is not connected or len is too long**
E_PAR	**A parameter erro or len is invalid**

Description This discards the data of length len in the buffer, returned by tcp_rcv_buf, of the TCP Communication End Point specified by cepid. There is no Wait state in this service call. Please make sure to release the buffer extracted from this service call.

tcp_snd_oob

Function Transmission of urgent data

Form **ER tcp_snd_oob(ID cepid, VP data, INT len, TMO tmout);**

Description This is not supported in current version.

tcp_rcv_oob

Function Reception of urgent data

Form **ER tcp_rcv_oob(ID cepid, VP data, INT len);**

Description This is not supported in current version.

tcp_can_cep

Function Cancellation of TCP processes

Form **ER tcp_can_cep(ID cepid, FN fncd);**

	cepid	**TCP Communication End Point ID**
	fncd	**Function code of the service call to be cancelled**

Return Value	**E_OK**	**Normal end**
	E_ID	**Incorrect ID number**
	E_NOEXS	**Non-existing TCP Communication End Point**
	E_PAR	**Parameter error (Incorrect fncd)**
	E_OBJ	**Process specified in fncd is not pending**

Description It cancels the execution of process that is pending in TCP Communication End Point specified by cepid. E_RLWI error is returned to the task that has issued the service call of the cancelled process. Or, when a nonblocking call is cancelled, the callback routine is called to notify the cancellation.

Please specify the process to be cancelled using the following Function codes. If TFN_TCP_ALL (= 0) is specified, all pending process can be cancelled.

TFN_TCP_ACP_CEP	Cancel the process of tcp_acp_cep
TFN_TCP_CON_CEP	Cancel the process of tcp_con_cep
TFN_TCP_CLS_CEP	Cancel the process of tcp_cls_cep
TFN_TCP_SND_DAT	Cancel the process of tcp_snd_dat
TFN_TCP_RCV_DAT	Cancel the process of tcp_rcv_dat
TFN_TCP_GET_BUF	Cancel the process of tcp_get_buf
TFN_TCP_RCV_BUF	Cancel the process of tcp_rcv_buf
TFN_TCP_SND_OOB	Cancel the process of tcp_snd_oob
TFN_TCP_ALL	Cancell all processes

tcp_set_opt

Function Setup of TCP Communication End Point Options

Form **ER tcp_set_opt(ID cepid, INT optname, VP optval, INT optlen);**

Description This is not supported in the current version.

tcp_get_opt

Function Get TCP Communication End Point Options

Form **ER tcp_get_opt(ID cepid, INT optname, VP optval, INT optlen);**

Description This is not supported in the current version.

Chapter 7 UDP Service Call

List of UDP Service Calls

udp_cre_cep	Creation of UDP Communication End Point
udp_vcre_cep	Creation of UDP Communication End Point(Automatic ID assignement)
udp_del_cep	Deletion of UDP Communication End Point
udp_snd_dat	Packet transmission
udp_rcv_dat	Packet reception
udp_can_cep	Cancellation of UDP processes
udp_set_opt	Setup of UDP Communication End Point Options
udp_get_opt	Get UDP Communication End Point Options

udp_cre_cep

Function Creation of UDP Communication End Point

Form **ER udp_cre_cep(ID cepid, T_UDP_CCEP *pk_ccep);**

cepid **UDP Communication End Point ID**

pk_ccep **Pointer to the structure for information to create UDP Communication End Point**

```
typedef struct t_udp_ccep {
    ATR cepatr;          UDP Communication End Point attributes (Unused, 0)
    T_IPV4EP myaddr;     Local IP Address and Port Number
    FP callback;         Callback routine
} T_UDP_CCEP;

typedef struct {
    UW ipaddr;           IP Address
    UH portno;           Port Number
} T_IPV4EP;
```

Return Value

E_OK **Normal end**

E_ID **Incorrect ID Number**

E_OBJ **UDP Communication End Point already created or Port Number already in use**

Description UDP Communication End Point specified by cepid is created. Reception is possible after creation of UDP Communication End Point only. If an UDP packet is received before the udp_rcv_dat service call is issued, reception is notified by callback.

Maximum queuing limit of packets to send and reception requests is decided by configuration. Transmission and request exceeding the queuing limit are ignored.

When IPV4_ADDRANY (=0) is specified as local address, the protocol stack uses the value of default_ipaddr. When UDP_PORTANY (=0) is specified as port number, protocol stack sets an arbitrary value for it.

udp_vcre_cep

Function Creation of UDP Communication End Point (Autimatic ID assignment)

Form **ER udp_vcre_cep(T_UDP_CCEP *pk_ccep);**

pk_ccep Pointer to the structure with information for creation of UDP Communication End Point

Return Value

E_OK	**Normal end**
E_ID	**UDP Communication End Point ID unavailable**
E_OBJ	**Port Number is already used**

Description It searched for unused UCP Communication End Point ID and allocates it. If UDP Communication End Point ID cannot be assigned, E_ID error is returned. Apart from this, it is same as udp_cre_cep.

Supplement This system call is ecTCP/IP Original.

udp_del_cep

Function Delection of UDP Communication End Point

Form **ER udp_del_cep(ID cepid);**

Cepid UDP Communication End Point ID

Return Value

E_OK Normal end

E_ID Incorrect ID number

E_NOEXS Non-existing UDP Communication End Point

Description UDP Communication End Point specified by cepid is deleted. If there is any task that has issued the service call for UDP transmission or reception for the deleted UDP Communication End Point and is in Wait state, the wait is released and E_DLT error is returned. The packets queued for transmission/reception are dumped.

udp_snd_dat

Function Packet transmission

Form **ER udp_snd_dat(ID cepid, T_IPV4EP *p_dstaddr, VP data, INT len, TMO tmout);**

cepid **UDP Communication End Point ID**

dstaddr **Pointer to the structure of remote IP Address/Port Number**

data **Pointer to packet for transmission**

len **Length of packet for transmission**

tmout **Timeout specification**

```
typedef struct {
    UW ipaddr;      IP Address
    UH portno;      Port Number
} T_IPV4EP;
```

Return Value

Positive Value **Normal end (Length of data copied into the Send buffer)**

E_ID **Incorrect ID number**

E_NOEXS **Non-existing UDP Communication End Point**

E_QOVR **Queue overflow**

E_DLT **UDP Communication End Point deleted while waiting for completion of transmission**

E_WBLK **Nonblocking call received**

E_TMOUT **Polling failed or timed out**

E_RLWAI **Process cancelled, Wait state released forcibly or no answer from destination**

E_PAR **Pointer to the send packet is set to odd memory address.**

Description From the UDP Communication End Point specified by cepid, it transmits the packet pointed by “data”, of the length “len”, to the remote specified by *dstaddr. In ecTCP/IP, there is no internal Send buffer for UDP in the protocol stack. The issuing task of this service call will be in Wait state until the packet is copied to the buffer memory of the device.

When this service call is issued with no-timeout specification (tmout = TMO_FEVR), the issuing task will be in Wait state until the copy of data to the buffer is completed.

When this service call is issued with a timeout value (tmout = 1～0x7fffffff), E_TMOUT error is returned when the specified time passed and buffer memory is not empty.

When this service call is issued in polling mode (tmout = TMO_POL) and there is no space in the buffer memory, E_TMOUT error is returned immediately.

When this service call is issued as nonblocking (tmout = TMO_NBLK) and there is no space in the buffer memory, the service call is returned immediately and the completion of data copy to the buffer memory is notified by the callback.

It is possible to queue the transmission of packets. Hence, two or more service calls can be issued for the same UDP Communication End Point. However, the memory area pointed by “data” cannot be used until the copy to buffer memory is completed. The maximum queuing limit for send packets is determined by the value specified in the macro UDP_QCNT. If udp_snd_dat us ussued exceeding the limit, E_QOVR error ir returned.

The send packet mentioned here, is the data part of the UDP packet. The UDP header is added to it inside the protocol stack.

Supplement The pointer of the send packet must be set to even memory address.

udp_rcv_dat

Function Packet reception

Form **ER udp_rcv_dat(ID cepid, T_IPV4EP *p_dstaddr, VP data, INT len,TMO tmout);**

cepid	**UDP Communication End point ID**
dstaddr	**Pointer to the structure of remote IP Address/Port Number**
data	**Pointer to the area where received packet is copied**
len	**The length of area where the received packet should be copied**
tmout	**Timeout specification**

```
typedef struct {
    UW ipaddr;      IP Address
    UH portno;      Port Number
} T_IPV4EP;
```

Return Value

Positive Value	**Normal end (Length of data copied)**
E_ID	**Incorrect ID number**
E_NOEXS	**Non-existing UDP Communication End Point**
E_DLT	**UDP Communication End Point was deleted while waiting for reception**
E_WBLK	**Nonblocking call received**
E_TMOUT	**Polling failed or timed out**
E_RLWAI	**Process cancelled or Wait state released forcibly**
E_PAR	**The length of the area where the received packet is to be copied is small or the pointer to the area is set to odd memory address**

Description For UDP Communication End Point specified by cepid, the packet is received to the area pointed by “data”. The length of the received packet is returned as return value.

Remote IP Address and Port Number are returned in *dstaddr.

When the length len of the area where received packet is to be copied, is shorter than the length of the received packet, data is copied till the area gets full, rest of the data is dumped and E_BOVR error is returned.

In case the packet is not received, the issuing task of this service call will be in Wait state

until a packet is received.

When this service call is issued with no-timeout specification (tmout = TMO_FEVR), the issuing task will be in Wait state until a packet is received.

When this service call is issued with a timeout value (tmout = 1～0x7fffffff), E_TMOUT error is returned when the specified time passes and no packet is received.

This service call cannot be issued in polling mode (tmout = TMO_POL) except from a callback function. In that case, E_PAR error is returned.

When this service call is issued as nonblocking (tmout = TMO_NBLK) and there is no received packet, the service call returns immediately and the completion of packet reception is notified by the callback.

The reception request of this service call can be queued. Hence, udp_rcv_dat can be issued two or more times simultaneously for the same UDP Communication End Point. If udp_rcv_dat is issued exceeding a maximum limit, E_QOVR error is returned. The received packet mentioned here is the data part of the UDP packet. The UDP header is removed inside the protocol stack.

Supplement For the buffer to be useful for holding data in the protocol stack, it must be bigger than 20 bytes. Moreover, when two or more tasks receive using this end point, data is received in the queued order. In this case, please do not specify the same memory area as Receive buffer.

The pointer of the Receive buffer must be even address in the memory.

When this service call is issued as nonblocking (tmout = TMO_NBLK), please do not use stack area for the pointer to the structure containing remote IP Address/Port Number (dstaddr) because writing to this area is done after connection.

udp_can_cep

Function Cancellation of UDP processes

Form **ER udp_can_cep(ID cepid, FN fncd);**

	cepid	**UDP Communication End Point ID**
	fncd	**Function code for the service call to be cancelled**

Return Value

E_OK	**Normal End**
E_ID	**Incorrect ID number**
E_NOEXS	**Non-exiting UDP Communication End Point**
E_PAR	**Parameter error (Incorrect fncd)**
E_OBJ	**The process specified by fncd is not pending**

Description The execution of the pending process in the UDP Communication End Point specified by cepid is cancelled. E_RLWAI error is returned to the task that had issued the service call for the cancelled processing. Or, when a nonblocking call is cancelled, the callback routine is called to notify it.

Please specify the process to be cancelled using the following functional code. If TFN_UDP_ALL (=0) is specified, all processings are cancelled.

TFN_UDP_SND_DAT	**Cancel the processing of udp_snd_dat**
TFN_UDP_RCV_DAT	**Cancel the processing of udp_rcv_dat**
TFN_UDP_ALL	**All processings are cancelled**

udp_set_opt

Function Setup the UDP Communication End Point Options

Form **ER udp_set_opt(ID cepid, INT optname, VP optval, INT optlen);**

cepid	**UDP Communication End Point ID**
optname	**Type of option**
optval	**Pointer to the buffer that contains option value**
optlen	**Length of option value**

Following can be used for optname.

IP_BROADCAST

Permission for transmission and reception of broadcast packets

Type BOOL optval TRUE:Permitted/FALSE:Refused

IP_ADD_MEMBERSHIP

Join a multicast group

Type UW optval Multicast address

IP_DROP_MEMBERSHIP

Leave a multicast group

Type UW optval Multicast address

Return Value

E_OK	**Normal end**
E_ID	**Incorrect ID number**
E_NOEXS	**Non-existing UDP Communication End Point**
E_PAR	**Invalid option type**
E_OBJ	**Value in optval is invalid**
E_NOSPT	**Function is not supported**

Example About IP broadcast packet transmission and reception

To transmit and receive a broadcasting packet, udp_set_opt function is used to confirm the broadcast transmission and reception. Transmission to IP Address 255.255.255.255 (limited broadcast) is sent.

```
T_UDP_CCEP udp_cep = {0, {IPV4_ADDRANY, UDP_PORTANY}, NULL};
BOOL optval;
```

```
/* Create UDP Communication End Point */
udp_cre_cep(cepid, &udp_cep);

/* Enable transmission and reception of broadcast */
udp_set_opt(cepid, IP_BROADCAST, (VP)TRUE, sizeof(BOOL));

/* Send broadcast packet */
dstaddr.ipaddr = 0xFFFFFFFF;
dstaddr.portno = UDP_PORT_NO;
udp_snd_dat(cepid, &dstaddr, &pkt, sizeof(pkt), TMO_FEVR);
```

About transmission and reception of IP multicast packet

Multicasting supprots transmission and reception of class D IP address and IGMP. For transmission, plain multicast address of the destination is specified. For reception, it is required to join the group of multicast address.

```
T_UDP_CCEP udp_cep = {0, {IPV4_ADDRANY, UDP_PORTANY}, NULL};
UB ipaddr[] = { 225, 6, 7, 8 }; /* Multicast address */
T_IPV4EP addr;

/* Creation of UDP Communication End Point */
udp_cre_cep(cepid, &udp_cep);

/* Transmission of multicast address */
dstaddr.ipaddr = byte4_to_long(ipaddr);
dstaddr.portno = UDP_PORT_NO;
udp_snd_dat(cepid, &dstaddr, &pkt, sizeof(pkt), TMO_FEVR);

/* Join group for multicast address 225.6.7.8 */
addr.ipaddr = byte4_to_long(ipaddr);
addr.portno = 1100;
udp_set_opt(cepid, IP_ADD_MEMBERSHIP, (VP)&addr, sizeof(mreq));

/* Reception of multicast packet */
udp_rcv_dat(cepid, &dstaddr, &pkt, sizeof(pkt), TMO_FEVR);

/* Leave group for multicast address 225.6.7.8 */
udp_set_opt(cepid, IP_DROP_MEMBERSHIP, (VP)&addr, sizeof(mreq));
```

*For transmission and reception of multicast packet, the device driver needs to be setup for tranmission and reception of multicast packet.

udp_get_opt

Function Get UDP Communication End Point Option

Form **ER udp_get_opt(ID cepid, INT optname, VP optval, INT optlen);**

Description This function is not supported in the current version.

Chapter 8 Callback

Completion of a nonblocking call

Function Notification for completion of a nonblocking call

Form **ER callback(ID cepid, FN fncd, VP parblk);**

cepid	**TCP or UDP Communicaton End Point ID**
fncd	**The function code of the completed service call**
parblk	**Pointer to the location where the error code is saved**

Return Value **Unused (0)**

Description This is called from protocol stack when processing of a nonblocking call is completed or cancelled.

Please access parblk by casting it as ER ercd = *(ER *) parblk. The error code returned from the service call is stored at this place.

In the above mentioned form, the expression “callback”is the callback routine defined by the user at the time of TCP Communication End Point creation and the function name is arbitrary.

TFN_TCP_ACP_CEP (-0x205) Notice for completion of tcp_acp_cep
TFN_TCP_CON_CEP (-0x206) Notice for completion of tcp_con_cep
TFN_TCP_CLS_CEP (-0x208) Notice for completion of tcp_cls_cep
TFN_TCP_SND_DAT (-0x209) Notice for completion of tcp_snd_dat
TFN_TCP_RCV_DAT (-0x20a) Notice for completion of tcp_rcv_dat
TFN_TCP_GET_BUF (-0x20b) Notice for completion of tcp_get_buf
TFN_TCP_RCV_BUF (-0x20d) Notice for completion of tcp_rcv_buf

TFN_UDP_RCV_DAT (-0x223) Notice for completion of udp_rcv_dat
TFN_UDP_SND_DAT (-0x224) Notice for completion of udp_snd_dat

Reception of urgent data

Function Notice of reception of urgent data

Form **ER callback(ID cepid, FN fncd, VP parblk);**

cepid **TCP Communication End Point ID**

fncd **TEV_TCP_RCV_OOB only**

parblk **Pointer to the location where the length of the urgent datga is stored**

Return Value **Unused (0)**

Description This is called when urgent data is received. In the callback rountine, urgent data must be extracted using tcp_rcv_oob. If the callback rountine returns without extracting the data, urgent data is dumped.

Please access parblk by casting it as INT len = *(INT *) parblk. Length of the urgent data is stored at this place.

In the above mentioned form, the expression callback is the same as the function for notification of the completion of nonblocking call.

* tcp_rcv_oob is not supported in the current version.

Reception of UDP packet

Function Notice of reception of UDP packet

Form **ER callback(ID cepid, FN fncd, VP parblk);**

cepid	**UDP Communication End Point ID**
fncd	**TEV_UDP_RCV_DAT only**
parblk	**Pointer to the location where the length of the packet is stored**

Return Value **Unused (0)**

Description This is called when a UDP packet is received and there is no udp_rcv_dat pending, i.e. when no UDP reception request is queued. In this callback routine, it is necessary to extract the received packet using udp_rcv_dat. The received packet is dumped if the callback returns without extracting the data.

Please access parblk by casting it as INT len = *(INT *) parblk. The length of the received packet is stored at this place.

In the above mentioned form, the callback expression is the callback routine defined by the user at the time of UDP Communication End Point creation and the function name is arbitrary.

Chapter 9 Original system functions

Protocol stack initialization

Function	Protocol stack initialization	
Form	**ER tcp_ini();**	
Return Value	**E_OK**	**Normal end**
	Negative Value	**Creation of internal OS resources failed**
Description	Creates resources and initializes the data to be used inside the protocol stack. This must be called once from the user task before calling other service calls.	

ecRTOS

IEEE 2050-2018 compliant Real Time OS

www.ecrtos.com

General inquiry / Technical support request: support@ecrtos.com