

# R-IN32 Series User's Manual

( $\mu$ Net3/BSD edition)

- R-IN32M3-EC
- R-IN32M3-CL
- R-IN32M4-CL2

All information contained in these materials, including products and product specifications, represents information on the product at the time of publication and is subject to change by Renesas Electronics Corp. without notice. Please review the latest information published by Renesas Electronics Corp. through various means, including the Renesas Electronics Corp. website (<http://www.renesas.com>)

Document number: R18UZ0064EJ0100

Issue date: Sep 5, 2016

Renesas Electronics

[www.renesas.com](http://www.renesas.com)



## Notice

1. Descriptions of circuits, software and other related information in this document are provided only to illustrate the operation of semiconductor products and application examples. You are fully responsible for the incorporation of these circuits, software, and information in the design of your equipment. Renesas Electronics assumes no responsibility for any losses incurred by you or third parties arising from the use of these circuits, software, or information.
2. Renesas Electronics has used reasonable care in preparing the information included in this document, but Renesas Electronics does not warrant that such information is error free. Renesas Electronics assumes no liability whatsoever for any damages incurred by you resulting from errors in or omissions from the information included herein.
3. Renesas Electronics does not assume any liability for infringement of patents, copyrights, or other intellectual property rights of third parties by or arising from the use of Renesas Electronics products or technical information described in this document. No license, express, implied or otherwise, is granted hereby under any patents, copyrights or other intellectual property rights of Renesas Electronics or others.
4. You should not alter, modify, copy, or otherwise misappropriate any Renesas Electronics product, whether in whole or in part. Renesas Electronics assumes no responsibility for any losses incurred by you or third parties arising from such alteration, modification, copy or otherwise misappropriation of Renesas Electronics product.
5. Renesas Electronics products are classified according to the following two quality grades: "Standard" and "High Quality". The recommended applications for each Renesas Electronics product depends on the product's quality grade, as indicated below.

"Standard": Computers; office equipment; communications equipment; test and measurement equipment; audio and visual equipment; home electronic appliances; machine tools; personal electronic equipment; and industrial robots etc.

"High Quality": Transportation equipment (automobiles, trains, ships, etc.); traffic control systems; anti-disaster systems; anti-crime systems; and safety equipment etc.

Renesas Electronics products are neither intended nor authorized for use in products or systems that may pose a direct threat to human life or bodily injury (artificial life support devices or systems, surgical implantations etc.), or may cause serious property damages (nuclear reactor control systems, military equipment etc.). You must check the quality grade of each Renesas Electronics product before using it in a particular application. You may not use any Renesas Electronics product for any application for which it is not intended. Renesas Electronics shall not be in any way liable for any damages or losses incurred by you or third parties arising from the use of any Renesas Electronics product for which the product is not intended by Renesas Electronics.

6. You should use the Renesas Electronics products described in this document within the range specified by Renesas Electronics, especially with respect to the maximum rating, operating supply voltage range, movement power voltage range, heat radiation characteristics, installation and other product characteristics. Renesas Electronics shall have no liability for malfunctions or damages arising out of the use of Renesas Electronics products beyond such specified ranges.
7. Although Renesas Electronics endeavors to improve the quality and reliability of its products, semiconductor products have specific characteristics such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Further, Renesas Electronics products are not subject to radiation resistance design. Please be sure to implement safety measures to guard them against the possibility of physical injury, and injury or damage caused by fire in the event of the failure of a Renesas Electronics product, such as safety design for hardware and software including but not limited to redundancy, fire control and malfunction prevention, appropriate treatment for aging degradation or any other appropriate measures. Because the evaluation of microcomputer software alone is very difficult, please evaluate the safety of the final products or systems manufactured by you.
8. Please contact a Renesas Electronics sales office for details as to environmental matters such as the environmental compatibility of each Renesas Electronics product. Please use Renesas Electronics products in compliance with all applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive. Renesas Electronics assumes no liability for damages or losses occurring as a result of your noncompliance with applicable laws and regulations.
9. Renesas Electronics products and technology may not be used for or incorporated into any products or systems whose manufacture, use, or sale is prohibited under any applicable domestic or foreign laws or regulations. You should not use Renesas Electronics products or technology described in this document for any purpose relating to military applications or use by the military, including but not limited to the development of weapons of mass destruction. When exporting the Renesas Electronics products or technology described in this document, you should comply with the applicable export control laws and regulations and follow the procedures required by such laws and regulations.
10. It is the responsibility of the buyer or distributor of Renesas Electronics products, who distributes, disposes of, or otherwise places the product with a third party, to notify such third party in advance of the contents and conditions set forth in this document. Renesas Electronics assumes no responsibility for any losses incurred by you or third parties as a result of unauthorized use of Renesas Electronics products.
11. This document may not be reproduced or duplicated in any form, in whole or in part, without prior written consent of Renesas Electronics.
12. Please contact a Renesas Electronics sales office if you have any questions regarding the information contained in this document or Renesas Electronics products, or if you have any other inquiries.

(Note 1) "Renesas Electronics" as used in this document means Renesas Electronics Corporation and also includes its majority-owned subsidiaries.

(Note 2) "Renesas Electronics product(s)" means any product developed or manufactured by or for Renesas Electronics.

## Instructions for the use of product

In this section, the precautions are described for over whole of CMOS device.

Please refer to this manual about individual precaution.

When there is a mention unlike the text of this manual, a mention of the text takes first priority.

### 1. Handling of Unused Pins

Handle unused pins in accord with the directions given under Handling of Unused Pins in the manual.

- The input pins of CMOS products are generally in the high-impedance state. In operation with an unused pin in the open-circuit state, extra electromagnetic noise is induced in the vicinity of LSI, associated shoot-through current flows internally, and malfunctions occur due to the false recognition of the pin state as an input signal become possible. Unused pins should be handled as described under Handling of Unused Pins in the manual.

### 2. Processing at Power-on

The state of the product is undefined at the moment when power is supplied.

- The states of internal circuits in the LSI are indeterminate and the states of register settings and pins are undefined at the moment when power is supplied.

In a finished product where the reset signal is applied to the external reset pin, the states of pins are not guaranteed from the moment when power is supplied until the reset process is completed.

In a similar way, the states of pins in a product that is reset by an on-chip power-on reset function are not guaranteed from the moment when power is supplied until the power reaches the level at which resetting has been specified.

### 3. Prohibition of Access to Reserved Addresses

Access to reserved addresses is prohibited.

- The reserved addresses are provided for the possible future expansion of functions. Do not access these addresses; the correct operation of LSI is not guaranteed if they are accessed.

### 4. Clock Signals

After applying a reset, only release the reset line after the operating clock signal has become stable. When switching the clock signal during program execution, wait until the target clock signal has stabilized.

- When the clock signal is generated with an external resonator (or from an external oscillator) during a reset, ensure that the reset line is only released after full stabilization of the clock signal. Moreover, when switching to a clock signal produced with an external resonator (or by an external oscillator) while program execution is in progress, wait until the target clock signal is stable.

- ARM, AMBA, ARM Cortex, Thumb, ARM Cortex-M3 and Cortex-M4F are a trademark or a registered trademark of ARM Limited in EU and other countries.
- Ethernet is a registered trademark of Fuji Xerox Limited.
- IEEE is a registered trademark of the Institute of Electrical and Electronics Engineers, Inc.
- EtherCAT is a registered trademark of Beckhoff Automation GmbH, Germany.
- CC-Link and CC-Link IE Field are a registered trademark of CC-Link Partner Association (CLPA).
- Additionally all product names and service names in this document are a trademark or a registered trademark which belongs to the respective owners.

# How to Use This Manual

## 1. Purpose and Target Readers

This manual is intended for users who wish to understand the functions of an Ethernet communication LSI "R-IN32M4-CL2" for designing application of it. It is assumed that the reader of this manual has general knowledge in the fields of electrical engineering, logic circuits, and microcontrollers.

Particular attention should be paid to the precautionary notes when using the manual. These notes occur within the body of the text, at the end of each section, and in the Usage Notes section.

The revision history summarizes the locations of revisions and additions. It does not list all revisions. Refer to the text of the manual for details.

The mark "<R>" means the updated point in this revision. The mark "<R>" let users search for the updated point in this document.

**Literature** Literature may be preliminary versions. Note, however, that the following descriptions do not indicate "Preliminary". Some documents on cores were created when they were planned or still under development. So, they may be directed to specific customers. Last four digits of document number (described as \*\*\*\*) indicate version information of each document. Please download the latest document from our web site and refer to it.

The document related to R-IN32 Series

Document Name	Document Number
R-IN32M3 Series Datasheet	R18DS0008EJ****
R-IN32M3-EC User's Manual	R18UZ0003EJ****
R-IN32M3-CL User's Manual	R18UZ0005EJ****
R-IN32M3 Series User's Manual (Peripheral function)	R18UZ0007EJ****
R-IN32M3 Series Programming Manual (OS edition)	R18UZ0011EJ****
R-IN32M3 Series Programming Manual (Driver edition)	R18UZ0009EJ****
R-IN32M4-CL2 User's Manual	R18UZ0032EJ****
R-IN32M4-CL2 User's Manual (Peripheral Modules)	R18UZ0034EJ****
R-IN32M4-CL2 User's Manual (Gigabit Ethernet PHY edition)	R18UZ0044EJ****
R-IN32M4-CL2 Programming Manual (Driver edition)	R18UZ0036EJ****
R-IN32M4-CL2 Programming Manual (OS edition)	R18UZ0040EJ****
R-IN32 Series User's Manual (µNet3/BSD edition)	R18UZ0064EJ****

## 2. Notation of Numbers and Symbols

Weight in data notation: Left is high-order column, right is low-order column

Active low notation:

xxxZ (capital letter Z after pin name or signal name)  
or xxx\_N (capital letter \_N after pin name or signal name)  
or xxxn (pin name or signal name contains small letter n)

Note:

Explanation of (Note) in the text

Caution:

Item deserving extra attention

Remark:

Supplementary explanation to the text

Numeric notation:

Binary ... xxxx , xxxxB or n'bxxxx (n bits)

Decimal ... xxxx

Hexadecimal ... xxxxH or n'hxxxx (n bits)

Prefixes representing powers of 2 (address space, memory capacity):

K (kilo) ...  $2^{10} = 1024$

M (mega) ...  $2^{20} = 1024^2$

G (giga) ...  $2^{30} = 1024^3$

Data Type:

Word ... 32 bits

Halfword ... 16 bits

Byte ... 8 bits

# Contents

1.	Introduction .....	1
2.	Specification .....	2
2.1	Position in the POSIX Specification.....	2
2.2	Differences from the $\mu$ Net3 .....	2
2.3	Compatibility of Symbol Name .....	2
3.	Module Structure .....	3
3.1	Module Structure .....	3
3.2	Header Structure .....	4
3.3	Source Files .....	5
4.	Supported API .....	6
4.1	Supported API Functions .....	6
4.2	Detail for Individual API Functions .....	7
4.2.1	<i>socket</i> (Create an Endpoint for Communication).....	7
4.2.2	<i>bind</i> (Assign a Name to a Socket).....	8
4.2.3	<i>listen</i> (Waits for a Connection on a Socket) .....	9
4.2.4	<i>accept</i> (Accept a Connection on a Socket) .....	10
4.2.5	<i>connect</i> (Make a Connection on a Socket).....	11
4.2.6	<i>send</i> (Transmit a Message to a Socket).....	12
4.2.7	<i>sendto</i> (Transmit a Message to a Socket).....	13
4.2.8	<i>recv</i> (Receive a Message from a Socket).....	14
4.2.9	<i>recvfrom</i> (Receive a Message from a Socket).....	15
4.2.10	<i>shutdown</i> (Cause Parts of a Full-Duplex Connection on the Socket to be Shut Down).....	16
4.2.11	<i>close</i> (Close a Socket).....	17
4.2.12	<i>select</i> (Synchronous I/O Multiplexing).....	18
4.2.13	<i>getsockname</i> (Retrieve the Name of a Socket) .....	19
4.2.14	<i>getpeername</i> (Retrieves the Name of the Peer Connected to a Socket) .....	20
4.2.15	<i>getsockopt</i> (Retrieves Options Associate with a Socket) .....	21
4.2.16	<i>setsockopt</i> (Manipulate Options Associated with a Socket) .....	22
4.2.17	<i>ioctl</i> (Control Hardware Devices (Sockets)).....	23
4.2.18	<i>inet_addr</i> (IP Address Handling Routine) .....	24
4.2.19	<i>inet_aton</i> (IP Address Handling Routine).....	25
4.2.20	<i>inet_ntoa</i> (IP Address Handling Routine).....	26

4.2.21	<i>if_nametoindex</i> (Map a Network Interface Name to its Corresponding Index) .....	27
4.2.22	<i>if_indextoname</i> (Map a Network Interface Name to its Corresponding Index) .....	28
4.2.23	<i>rresyport</i> (Acquire a Socket with a Port Bound to It).....	29
4.2.24	<i>getifaddrs</i> (Retrieve Interface Addresses).....	30
4.2.25	<i>freeifaddrs</i> (Free List of Interface Information).....	31
5.	Socket Options .....	32
5.1	List of Options .....	32
6.	Capabilities .....	33
6.1	Non-Blocking Mode .....	33
6.2	Loopback .....	34
6.3	Error Processing.....	34
6.4	List of errno .....	35
7.	Implementing BSD Application .....	37
7.1	Source Code.....	37
7.2	Include Path .....	37
7.3	Configuration.....	38
7.4	Defining Resources.....	39
7.5	Kernel Objects .....	40
7.6	Initialization .....	40
8.	Appendix.....	41
8.1	Supported Compilers .....	41
8.2	Sample Application.....	41
8.3	Restrictions on Compilers.....	41

## List of Figures

Figure 3.1	$\mu$ Net3/BSD Module Structure .....	3
Figure 3.2	$\mu$ Net3/BSD Source Files .....	5
Figure 7.1	$\mu$ Net3 Library .....	37
Figure 7.2	"Include" Paths for $\mu$ Net3 .....	37



# List of Tables

Table 3.1	List of Header Files.....	4
Table 4.1	List of API Functions.....	6
Table 5.1	List of Options .....	32
Table 6.1	Non-Blocking APIs.....	33

## 1. Introduction

The  $\mu$ Net3/BSD socket API provides a BSD interface for running BSD applications on the  $\mu$ Net3 TCP/IP stack. The stack and API allow seamless operation of socket applications from the Linux or BSD environments.

This document describes how to use the  $\mu$ Net3/BSD API and restrictions related to the product.

## 2. Specification

### 2.1 Position in the POSIX Specification

The  $\mu$ Net3/BSD socket API is equivalent to 4.4 BSD-Lite. See Section 5, Socket Options for the APIs supported in this document. Using the  $\mu$ Net3/BSD API allows applications to use both BSD-based socket APIs and  $\mu$ Net3-based APIs.

### 2.2 Differences from the $\mu$ Net3

The  $\mu$ Net3/BSD API provides the following functionality for existing  $\mu$ Net3 in addition to a POSIX-compliant socket API.

- Multiple calls of socket API functions
- A select() function
- Loopback address
- Multicast grouping by sockets
- Listen queue of TCP sockets
- Socket errors

### 2.3 Compatibility of Symbol Name

The API functions, structures, and macros provided in the  $\mu$ Net3/BSD API are given the unique prefix “`unet3_`” to avoid conflicts between symbols in the compiler environment.

The POSIX standard symbol names used in applications are replaced by those ones with the prefix unique to  $\mu$ Net3/BSD by including `sys/socket.h`. This allows the operation of applications using BSD sockets under  $\mu$ Net3/BSD without changing the files of source code.

In this document, the symbols are indicated in the POSIX standard notation for readability.

### 3. Module Structure

#### 3.1 Module Structure

Figure 3.1 shows the module blocks composing the μNet3/BSD.

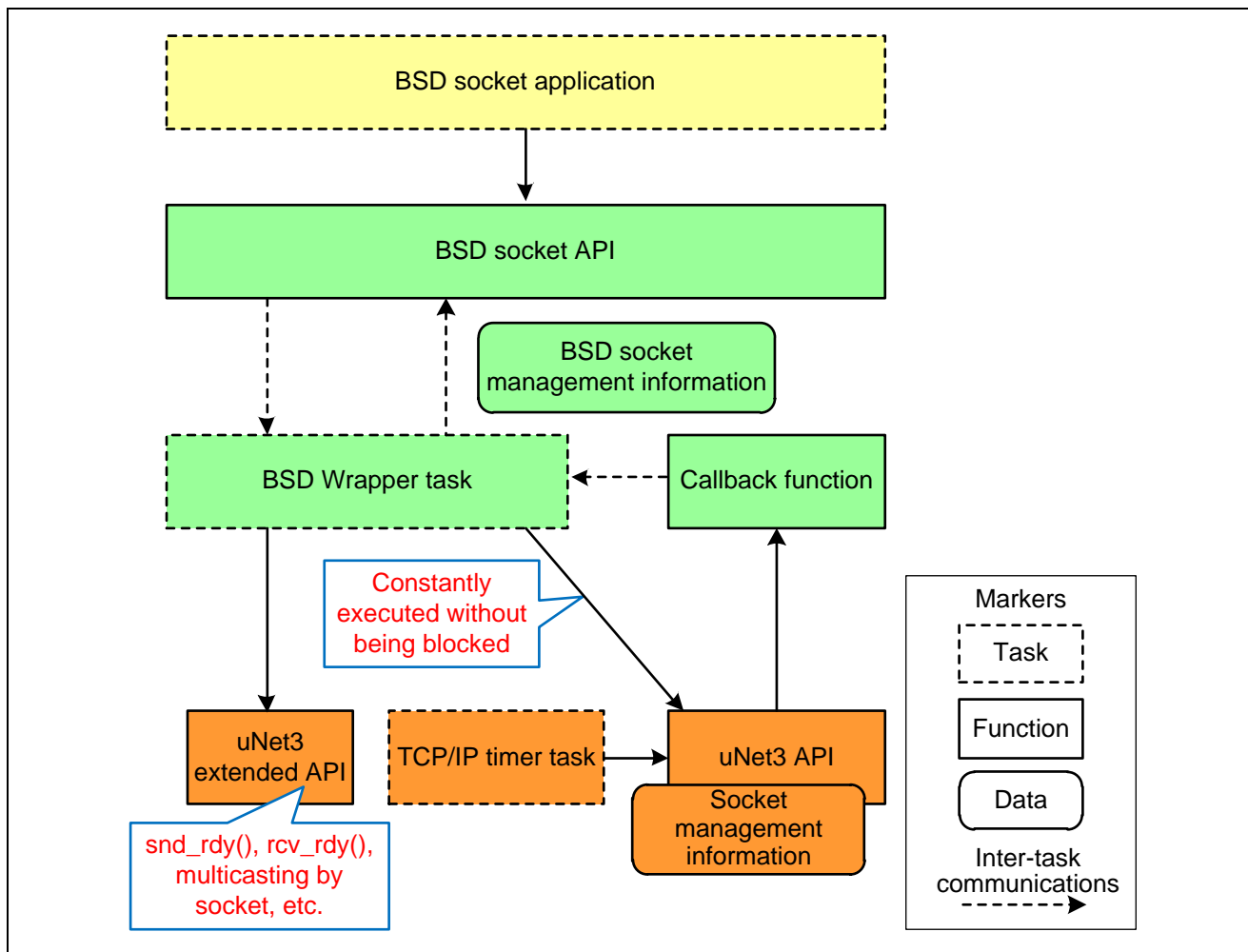


Figure 3.1 μNet3/BSD Module Structure

## 3.2 Header Structure

In the  $\mu$ Net3/BSD API, the POSIX-compliant header files are regarded as dummy files. These files include the open header file `unet3_socket.h`, an original file for  $\mu$ Net3/BSD. Table 3.1 lists the header files provided for the  $\mu$ Net3/BSD API.

Table 3.1 List of Header Files

Header File Name	Major Application
POSIX-Compliant Header Files (for Sockets)	
<code>arpa/inet.h</code>	Define the values for handling IP addresses
<code>netinet/in.h</code>	The address families <code>AF_INET</code> and <code>AF_INET6</code> which include IP addresses and TCP/UDP port numbers. It is widely used on the internet.
<code>netinet/ip.h</code>	Define the IP-level options and IP packets.
<code>netinet/tcp.h</code>	Define the TCP-level options and TCP packets.
<code>sys/socket.h</code>	This contains declarations of the core functions for the BSD sockets and their data structures.
<code>net/if.h</code>	Interface related definitions
POSIX-Compliant Header Files (for Systems)	
<code>sys/errno.h</code>	Definitions of error codes
<code>sys/ioctl.h</code>	<code>ioctl</code> related definitions
<code>sys/select.h</code>	Definitions of functions including <code>select</code> and <code>fd_set</code>
<code>sys/time.h</code>	Definitions of functions including <code>timeval</code> type
<code>sys/times.h</code>	Definitions of functions including <code>timeval</code> type
<code>sys/unistd.h</code>	Standard header related to the UNIX standard
$\mu$ Net3/BSD Original Header Files	
<code>unet3_cfg.h</code>	User-configuration definitions
<code>unet3_socket.h</code>	Open header file which defines the socket APIs
<code>unet3_sys.h</code>	Header file which defines types and macros specific to the BSD platform. The header files to be included for system integration of BSD applications are collected in this file.
<code>unet3_wrap.h</code>	For internal control
<code>bsd_param.h</code>	For internal control

### 3.3 Source Files

Source files used in the  $\mu$ Net3/BSD API are shown below.

When used in an application, \*.c programs under the bsd folder should be incorporated.

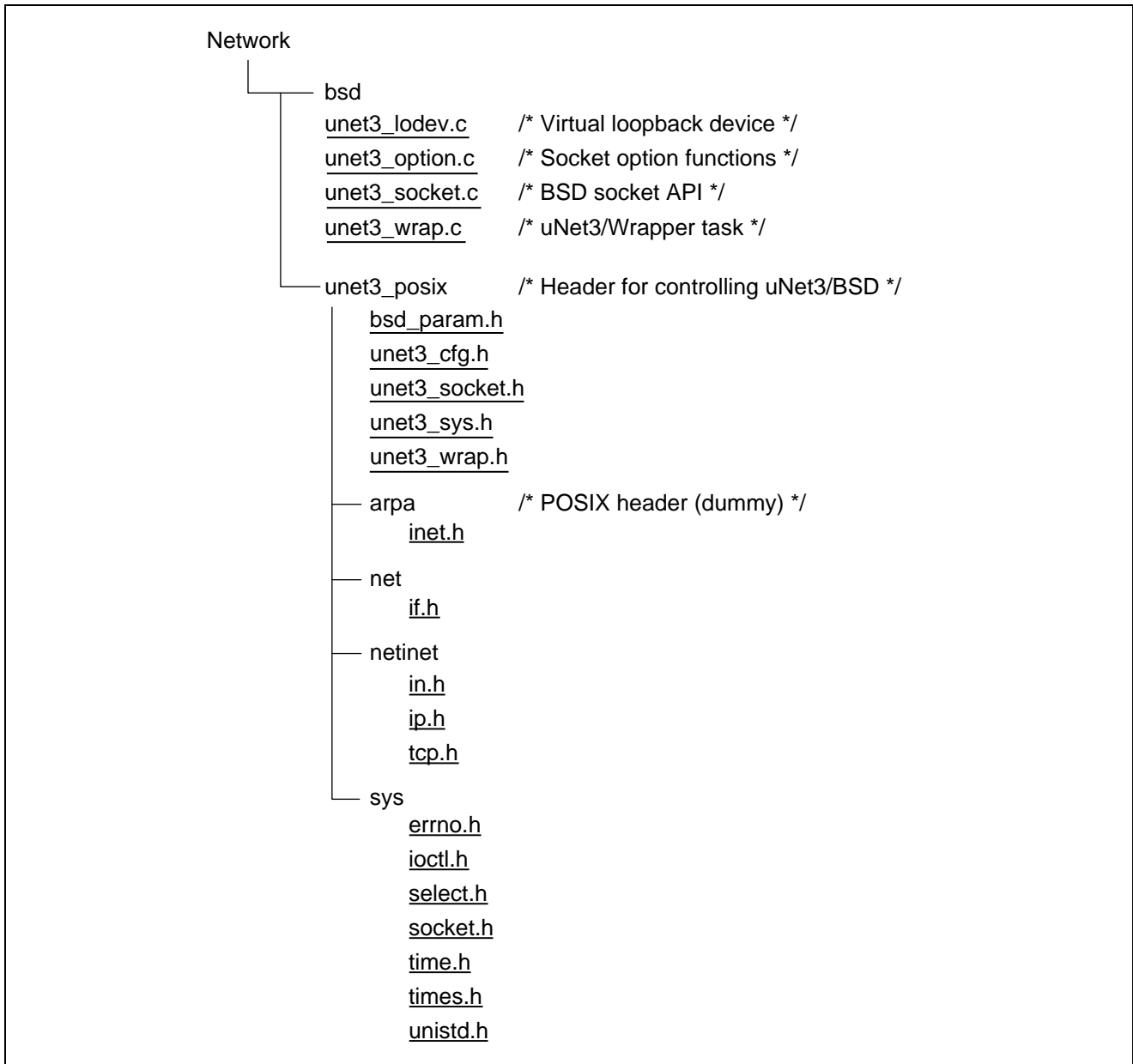


Figure 3.2  $\mu$ Net3/BSD Source Files

## 4. Supported API

### 4.1 Supported API Functions

Table 4.1 lists the API functions provided by the μNet3/BSD.

Table 4.1 List of API Functions

API Function	Description	Header for Inclusion
unet3_bsd_init	Initialize the μNet3/BSD	"sys/socket.h"
get_errno	Get the errnos for individual tasks	"sys/errno.h"
socket	Create an endpoint for communication	"sys/socket.h"
bind	Assign a name to a socket	"sys/socket.h"
listen	Waits for a connection on a socket	"sys/socket.h"
accept	Accept a connection on a socket	"sys/socket.h"
connect	Make a connection on a socket	"sys/socket.h"
send	Transmit a message to a socket	"sys/socket.h"
sendto	Transmit a message to a socket	"sys/socket.h"
recv	Receive a message from a socket	"sys/socket.h"
recvfrom	Receive a message from a socket	"sys/socket.h"
shutdown	Cause parts of a full-duplex connection on the socket to be shut down	"sys/socket.h"
close	Close a file descriptor (socket)	"sys/unistd.h"
select	Synchronous I/O multiplexing	"sys/select.h"
getsockname	Retrieve the name of a socket	"sys/socket.h"
getpeername	Retrieve the name of the peer connected to a socket	"sys/socket.h"
getsockopt	Retrieve options associated with a socket	"sys/socket.h"
setsockopt	Manipulate options associated with a socket	"sys/socket.h"
ioctl	Control hardware devices (sockets)	"sys/ioctl.h"
inet_addr	Internet address handling routine	"arpa/inet.h"
inet_aton	Internet address handling routine	"arpa/inet.h"
inet_ntoa	Internet address handling routine	"arpa/inet.h"
if_nametoindex	Map a network interface name to its corresponding index	"net/if.h"
if_indextoname	Map a network interface name to its corresponding index	"net/if.h"
resvport	Acquire a socket to which a port is bound	"sys/unistd.h"
getifaddrs	Retrieve the address of the interface	"sys/types.h"
freeifaddrs	Free the address of the interface	"sys/types.h"

## 4.2 Detail for Individual API Functions

### 4.2.1 *socket* (Create an Endpoint for Communication)

#### Format

```
#include "sys/socket.h"
int socket(int domain, int type, int protocol);
```

#### Parameters

int	domain	Domain
int	type	Communication type
int	protocol	Protocol

#### Returned value

int	Created socket FD. This function returns $-1$ on occurrence of an error.
-----	--

#### errno

ENOMEM	The number of sockets that can be created has been exceeded. Message buffer has been completely used up.
EINVAL	An invalid parameter was specified.

- Allowed domains are AF\_INET and AF\_INET6 only.
- Allowed communication types are SOCK\_STREAM and SOCK\_DGRAM only.
- Set any value for the protocol as it is not used in this function.
- The number of sockets that can be created at the same time (the sum of TCP and UDP) is the value defined by #define CFG\_NET\_SOC\_MAX.
- The number of TCP sockets that can be created at the same time is the value defined by #define CFG\_NET\_TCP\_MAX.
- Unlike in the POSIX specification, setting 0 as the local port of a socket is not allowed. A socket is assigned a temporary local port number immediately after it is created.



### 4.2.2 *bind* (Assign a Name to a Socket)

#### Format

```
#include "sys/socket.h"
int bind(int sockfd, const struct sockaddr *addr, unsigned int addrlen);
```

#### Parameters

int	sockfd	File descriptor of the socket
const struct sockaddr *	addr	Local address
unsigned int	addrlen	Local address length

#### Returned value

int	Result of processing. This function returns 0 on success and -1 if an error occurred.
-----	---

#### errno

EINVAL	An invalid parameter was specified.
ENOMEM	Message buffer has been completely used up.
EBADF	Invalid socket FD for binding
EAFNOSUPPORT	Unsupported address family
EADDRINUSE	Address already in use
EADDRNOTAVAIL	Cannot assign requested address.

- Local address should be set with the type struct sockaddr\_in.
- The only allowed addresses as the IP address (IPv4) for the local address are the one set for the device or INADDR\_ANY(unspecified).
- If the user sets PORT\_ANY(0) as the port number of the local address, a port number is assigned by the protocol stack.
- The only allowed local address length is sizeof(struct sockaddr\_in) (= 16).
- Set any value for the "sin\_len", a member of the type struct sockaddr\_in, as it is not used in this function.
- To start reception, including listening for incoming connections from TCP (listen()) and receiving UDP packets (recv(), recvfrom()), the user needs to specify the target socket and execute the bind() function in advance.
- Binding to the well-known port numbers (1 to 1023) is also allowed.

### 4.2.3 *listen* (Waits for a Connection on a Socket)

#### Format

```
#include "sys/socket.h"
int listen(int sockfd, int backlog);
```

#### Parameters

int	sockfd	File descriptor of the socket
int	backlog	Backlog

#### Returned value

int	Result of processing. This function returns 0 on success and -1 if an error occurred.
-----	---

#### errno

EINVAL	An invalid parameter was specified.
ENOMEM	Message buffer has been completely used up.
EBADF	Invalid socket FD for listening to
EPROTONOSUPPORT	Unsupported protocol (non-TCP socket)

- This function makes the TCP socket listen for an incoming connection.
- Allowed file descriptors are those ones for TCP sockets.
- The maximum number of back logs is defined by #define CFG\_NET\_TCP\_MAX -1.

#### 4.2.4 *accept* (Accept a Connection on a Socket)

##### Format

```
#include "sys/socket.h"
int accept(int sockfd, struct sockaddr *addr, unsigned int *addrlen);
```

##### Parameters

int	sockfd	File descriptor of the socket
struct sockaddr *	addr	Remote address (output)
unsigned int *	addrlen	Remote address length (output)

##### Returned value

int	The connected socket FD. This function returns -1 if an error occurred.
-----	---

##### errno

EINVAL	An invalid parameter was specified.
ENOMEM	Message buffer has been completely used up.
EBADF	The program is not listening to the specified socket.
EAGAIN	No connections have been made (in asynchronous network).
ETIMEDOUT	Connection attempt timed out (when a timeout is set).

- Allowed file descriptors are those ones for TCP sockets for which the listen() function succeeded.
- The remote address is set with the type struct sockaddr\_in\*.
- If no connections were established, this function blocks further processing until an attempt of connection from a remote party.

### 4.2.5 *connect* (Make a Connection on a Socket)

#### Format

```
#include "sys/socket.h"
int connect(int sockfd, const struct sockaddr *addr, unsigned int addrlen);
```

#### Parameters

int	sockfd	File descriptor of the socket
const struct sockaddr *	addr	Remote address
unsigned int	addrlen	Remote address length

#### Returned value

int	Result of processing. This function returns 0 on success and -1 if an error occurred.
-----	---

#### errno

EINVAL	An invalid parameter was specified.
ENOMEM	Message buffer has been completely used up.
EBADF	An invalid socket FD to connect
ECONNREFUSED	Connection refused by server
EAFNOSUPPORT	Unsupported address family
EISCONN	The socket is already connected.
	Listening to the socket is currently in progress.
EALREADY	A connection request is already in progress.
EAGAIN	A connection request is in progress (in asynchronous network).
ETIMEDOUT	Connection attempt timed out (when a timeout is set).

- The function `connect()` operates and behaves differently depending on the protocol of the specified socket FD and the type of transfer.
- When connecting to a TCP socket, the μNet3/BSD API transmits the SYN signal to the address of the remote target and attempts connection to it. This only applies to TCP sockets other than those which are currently connected or for which waiting for a connection is in progress.
- In transmission through a UDP socket, the address of the remote target is regarded as the destination of transmission. If an address different from that of the remote target is set in the `sendto()` function, the given address is regarded as that of the destination for transmission.
- Setting `AF_UNSPEC` in the `sa_family` member of the remote address clears the setting mentioned above.
- The μNet3/BSD API differs from the POSIX specification in that it does not apply filtering of remote addresses in reception through UDP sockets.
- The μNet3/BSD API differs from the POSIX specification in that it cannot reissue a connection request for a TCP socket whose input and output are driven asynchronously, once a connection has been established by the `connect()` function. For example, if the ability to write to the target TCP socket has been ensured by `select()` after `EAGAIN` was returned in response to `connect()`, the session with the socket has been established, so the transmission and reception of data are possible.

## 4.2.6 *send* (Transmit a Message to a Socket)

### Format

```
#include "sys/socket.h"
int send(int sockfd, const void *buf, unsigned int len, int flags);
```

### Parameters

int	sockfd	File descriptor of the socket
const void *	buf	Source address of the data for transmission
unsigned int	len	Length of the data for transmission
int	flags	Flag

### Returned value

int	Number of bytes of the transmitted data. This function returns -1 if an error occurred.
-----	---

### errno

EINVAL	An invalid parameter was specified.
ENOMEM	Message buffer has been completely used up. The size of the network buffer does not match the value set in len.
EBADF	An invalid socket FD for sending
EDESTADDRREQ	Destination address is not specified (UDP socket).
ENOTCONN	The socket is not connected (TCP socket).
EAGAIN	Transmission is in progress (in asynchronous network).
ETIMEDOUT	Connection attempt timed out (when a timeout is set).

- Valid values for the length of the data for transmission are between 1 and 65535.
- Set any value for the flag as it is not used in this function.
- Unlike in the POSIX specification, transmission of 0-byte UDP packet is not allowed.

### 4.2.7 *sendto* (Transmit a Message to a Socket)

#### Format

```
#include "sys/socket.h"
int sendto(int sockfd, const void *buf, unsigned int len, int flags, const struct sockaddr
*dest_addr, unsigned int addrlen);
```

#### Parameters

int	sockfd	File descriptor of the socket
const void *	buf	Source address of the data for transmission
unsigned int	len	Length of the data for transmission
int	flags	Flag
const struct sockaddr *	dest_addr	Address of the destination for transmission
unsigned int	addrlen	Size of the address of the destination

#### Returned value

int	Number of the bytes of the transmitted data. This function returns -1 if an error occurred.
-----	---

#### errno

EINVAL	An invalid parameter was specified.
ENOMEM	Message buffer has been completely used up. The size of the network buffer does not match the value set in len.
EBADF	An invalid socket FD for the sendto operation
EDESTADDRREQ	Destination address is not specified (UDP socket).
ENOTCONN	The socket is not connected (TCP socket).
EAGAIN	Transmission is in progress (in asynchronous network).
ETIMEDOUT	Connection attempt timed out (when a timeout is set).

- Valid values for the length of the data for transmission are between 1 and 65535.
- Set any value for the flag as it is not used in this function.
- The parameters for the address of the destination for transmission and the size of the address of the destination are not used when connecting to TCP sockets.
- Unlike in the POSIX specification, transmission of 0-byte UDP packet is not allowed.

## 4.2.8 *recv* (Receive a Message from a Socket)

### Format

```
#include "sys/socket.h"
int recv(int sockfd, void *buf, unsigned int len, int flags);
```

### Parameters

int	sockfd	File descriptor of the socket
void *	buf	Address of the reception buffer
unsigned int	len	Length of the reception buffer
int	flags	Flag

### Returned value

int	Number of the bytes of the received data including 0 byte. This function returns -1 if an error occurred.
-----	---

### errno

EINVAL	An invalid parameter was specified.
ENOMEM	Message buffer has been completely used up. The size of the network buffer does not match the value set in len.
EBADF	An invalid socket FD for the <i>recv</i> operation
ENOTCONN	The socket is not connected (TCP socket).
EAGAIN	No packet has been received (in asynchronous network).
ETIMEDOUT	Connection attempt timed out (when a timeout is set).

- Set any value for the flag as it is not used in this function.
- Valid values for the length of the reception data are between 1 and 65535.
- If no packets are received, this function blocks further processing until packet reception.
- This function returns an error if connection with a remote party has not been established.
- This function returns 0 if the TCP socket is disconnected from the remote party.

## 4.2.9 *recvfrom* (Receive a Message from a Socket)

### Format

```
#include "sys/socket.h"
int recvfrom(int sockfd, void *buf, unsigned int len, int flags, struct sockaddr *src_addr,
             unsigned int *addrlen);
```

### Parameters

int	sockfd	File descriptor of the socket
void *	buf	Reception buffer address
unsigned int	len	Reception buffer length
int	flags	Flag
struct sockaddr *	src_addr	Source address of the data for transmission
unsigned int *	addrlen	Size of the source address

### Returned value

int	Number of the bytes of the received data including 0 byte. This function returns -1 if an error occurred.
-----	---

### errno

EINVAL	An invalid parameter was specified.
ENOMEM	Message buffer has been completely used up. The size of the network buffer does not match the value set in len.
EBADF	An invalid socket FD for the <i>recvfrom</i> operation
ENOTCONN	The socket is not connected (TCP socket).
EAGAIN	No packet has been received (in asynchronous network).
ETIMEDOUT	Connection attempt timed out (when a timeout is set).

- Set any value for the flag as it is not used in this function.
- Valid values for the length of the reception data are between 1 and 65535.
- If no packets are received, this function blocks further processing until packet reception.
- This function returns an error if connection with a remote party has not been established.
- This function returns 0 if the TCP socket is disconnected from the remote party.
- The parameters for the source address of the data for transmission and the size of the source address are not used when connecting to TCP sockets.



#### 4.2.10 *shutdown* (Cause Parts of a Full-Duplex Connection on the Socket to be Shut Down)

##### Format

```
#include "sys/socket.h"
int shutdown(int sockfd, int how);
```

##### Parameters

int	sockfd	File descriptor of the socket
int	how	Type of shutdown

##### Returned value

int	Result of processing. This function returns 0 on success and -1 if an error occurred.
-----	---

##### errno

EINVAL	An invalid parameter was specified.
ENOMEM	Message buffer has been completely used up.
EBADF	An invalid socket FD to shutdown
EPIPE	The socket is not connected (TCP socket).

- The only allowed types of shutdown are SHUT\_WR and SHUT\_RDWR.

### 4.2.11 *close* (Close a Socket)

#### Format

```
#include "sys/unistd.h"
int close(int fd);
```

---

#### Parameters

int	fd	File descriptor of the socket
-----	----	-------------------------------

---

#### Returned value

int	Result of processing. This function returns 0 on success and -1 if an error occurred.
-----	---

---

#### errno

EINVAL	An invalid parameter was specified.
ENOMEM	Message buffer has been completely used up.
EBADF	An invalid socket FD to close

---

- If a TCP session is active when this function is called, the socket will be closed after the session has been cut off.
- Once the socket with the given FD is closed, it cannot be used again until a new connection is established.

### 4.2.12 *select* (Synchronous I/O Multiplexing)

#### Format

```
#include "sys/select.h"
int select(int nfd, fd_set *readfds, fd_set *writefds, fd_set *exceptfds, struct timeval *timeout);
```

#### Parameters

int	nfd	An integer one greater than the highest file descriptor in readfds and writefds. When adding file descriptors to either of the sets, increment this value by one.
fd_set *	readfds	A set of file descriptors to be checked for readability
fd_set *	writefds	A set of file descriptors to be checked for writability
fd_set *	exceptfds	A set of file descriptors to be checked for exceptional conditions (not supported)
struct timeval *	timeout	Time until expiration of the monitoring period

#### Returned value

int	The total number of file descriptors to be checked for readability or writability. This function returns 0 on timeout and -1 if an error occurred.
-----	--

#### errno

EINVAL	An invalid parameter was specified.
ENOMEM	Message buffer has been completely used up.
EBADF	The socket with the specified FD does not support select().

- The argument `exceptfds` is not used in this function.
- The μNet3/BSD API differs from the POSIX specification in that, when this function is executed for a file descriptor immediately after it has been created, the function allows writing if it is an UDP socket (reading is not possible if no packet has been received) and reading if it is a TCP socket (writing is not possible).

### 4.2.13 *getsockname* (Retrieve the Name of a Socket)

#### Format

```
#include "sys/socket.h"
int getsockname(int sockfd, struct sockaddr *addr, unsigned int *addrlen);
```

#### Parameters

int	sockfd	File descriptor of the socket
struct sockaddr *	addr	Pointer to the buffer where the socket address is stored
unsigned int *	addrlen	Size of the buffer where the socket address is stored

#### Returned value

int	Result of processing. This function returns 0 on success and -1 if an error occurred.
-----	---

#### errno

EINVAL	An invalid parameter was specified.
ENOMEM	Message buffer has been completely used up.
EBADF	An invalid socket FD for the <i>getsockname</i> operation

- The value in *\*addrlen* should be the size of *sockaddr\_in* (16 bytes or more).
- The address is bound to a socket when the following API functions are called.

```
bind()
connect()
accept()
send/sendto()
recv/recvfrom()
```

If a function from the above list fails, the value of the address associated with the socket will be undefined.

#### 4.2.14 *getpeername* (Retrieves the Name of the Peer Connected to a Socket)

##### Format

```
#include "sys/socket.h"
int getpeername(int sockfd, struct sockaddr *addr, unsigned int *addrlen);
```

##### Parameters

int	sockfd	File descriptor of the socket
struct sockaddr *	addr	Pointer to the buffer where the remote address is stored
unsigned int *	addrlen	Size of the buffer where the remote address is stored

##### Returned value

int	Result of processing. This function returns 0 on success and -1 if an error occurred.
-----	---

##### errno

EINVAL	An invalid parameter was specified.
ENOMEM	Message buffer has been completely used up.
EBADF	An invalid socket FD for the <i>getpeername</i> operation
ENOTCONN	Destination address required

- The value in *\*addrlen* should be the size of *sockaddr\_in* (16 bytes or more).
- For a TCP connection, this function only allows retrieval of the address of the remote party to which the TCP socket is connected.
- For a UDP connection, this function only allows retrieval of the address of a remote party with an address previously specified in a *connect* or *sendto* function, or of a socket which has received packets.

### 4.2.15 *getsockopt* (Retrieves Options Associate with a Socket)

#### Format

```
#include "sys/socket.h"
int getsockopt(int sockfd, int level, int optname, void *optval, unsigned int *optlen);
```

#### Parameters

int	sockfd	File descriptor of the socket
int	level	The level of the option
int	optname	Option name
void *	optval	A pointer to the buffer where the value of the option is to be stored
unsigned int *	optlen	The size of the buffer pointed to by optval

#### Returned value

int	Result of processing. This function returns 0 on success and -1 if an error occurred.
-----	---

#### errno

EINVAL	An invalid parameter was specified.
ENOMEM	Message buffer has been completely used up.
EBADF	An invalid socket FD for the <i>getsockopt</i> operation
EPROTONOSUPPORT	The option is not supported.

- Allowed option levels are SOL\_SOCKET, IPPROTO\_IP and IPPROTO\_TCP. Allowed option levels are SOL\_SOCKET, IPPROTO\_IP and IPPROTO\_TCP.
- The available option names at each option level are listed in the Section 5.1, List of Options.

### 4.2.16 *setsockopt* (Manipulate Options Associated with a Socket)

#### Format

```
#include "sys/socket.h"
int setsockopt(int sockfd, int level, int optname, const void *optval, unsigned int optlen);
```

#### Parameters

int	sockfd	File descriptor of the socket
int	level	The level of the option
int	optname	Option name
const void *	optval	The buffer in which the values of the requested options are to be returned
unsigned int	optlen	Size of the buffer pointed to by optval

#### Returned value

int	Result of processing. This function returns 0 on success and -1 if an error occurred.
-----	---

#### errno

EINVAL	An invalid parameter was specified.
ENOMEM	Message buffer has been completely used up.
EBADF	An invalid socket FD for the setsockopt operation
EPROTONOSUPPORT	The option is not supported.

- Allowed option levels are SOL\_SOCKET, IPPROTO\_IP and IPPROTO\_TCP.
- The available option names at each option level are listed in the Section 5.1, List of Options.

### 4.2.17 *ioctl* (Control Hardware Devices (Sockets))

#### Format

```
#include "sys/ioctl.h"
int ioctl(int d, int request, ...);
```

#### Parameters

int	d	File descriptor of the socket
int	request	Request
...		Parameter for the request

#### Returned value

int	Result of processing. This function returns 0 on success and -1 if an error occurred.
-----	---

#### errno

EINVAL	An invalid parameter was specified.
ENOMEM	Message buffer has been completely used up.
EFAULT	The parameter for the request is not usable.

- The only allowed request is FIONBIO (non-blocking mode).
- The parameters for requests for the non-blocking mode are 1 (set) and 0 (clear).
- See Section 6.1, Non-Blocking Mode for details on non-blocking mode.



#### 4.2.18 *inet\_addr* (IP Address Handling Routine)

##### Format

```
#include "arpa/inet.h"
unsigned int inet_addr(const char *cp);
```

---

##### Parameters

const char *	cp	The IP address in dot-notation
--------------	----	--------------------------------

---

##### Returned value

unsigned int	The IP address converted into binary data in network byte order
--------------	---

---

##### errno

Not specified
---------------

---

- The function returns 0 if conversion failed.

### 4.2.19 *inet\_aton* (IP Address Handling Routine)

#### Format

```
#include "arpa/inet.h"
int inet_aton(const char *cp, struct in_addr *inp);
```

#### Parameters

const char *	cp	The IP address in dot-notation
struct in_addr *	inp	Pointer to the buffer where the post-conversion binary value of the IP address is stored in network byte order

#### Returned value

int	Result of processing. This function returns 0 on success and -1 if an error occurred.
-----	---

#### errno

Not specified
---------------

- The function returns 0 if conversion failed.

#### 4.2.20 *inet\_ntoa* (IP Address Handling Routine)

##### Format

```
#include "arpa/inet.h"
char *inet_ntoa(struct in_addr in);
```

---

##### Parameters

struct in_addr	in	IP address as binary data in network byte order
----------------	----	---

---

##### Returned value

char *	The IP address converted into dot-notation
--------	--

---

##### errno

Not specified
---------------

---

- The string is returned in a statically allocated buffer in the area for the IP address converted into dot-notation, and will be overwritten by subsequent calls.

### 4.2.21 *if\_nametoindex* (Map a Network Interface Name to its Corresponding Index)

#### Format

```
#include "net/if.h"
unsigned int if_nametoindex(const char *ifname)
```

#### Parameters

const char *	ifname	Interface name
--------------	--------	----------------

#### Returned value

unsigned int	The index of the interface. This function returns 0 if an error occurred.
--------------	---

#### errno

ENXIO	An interface with the given name does not exist.
-------	--

- The setting for the name of the interface is based on the device name in μNet3 (gNET\_DEV[index-1].name[8]).

#### 4.2.22 *if\_indextoname* (Map a Network Interface Name to its Corresponding Index)

##### Format

```
#include "net/if.h"
char *if_indextoname(unsigned int ifindex, char *ifname)
```

##### Parameters

unsigned int	ifindex	Interface index
char *	ifname	Pointer to the buffer where the interface name is stored

##### Returned value

char *	Result of processing. This function returns ifname on success and null if an error occurred.
--------	--

##### errno

ENXIO	No index found for the interface.
-------	-----------------------------------

- The setting of the interface is based on the device name in μNet3 (gNET\_DEV[index-1].name[8]).

### 4.2.23 *rresvport* (Acquire a Socket with a Port Bound to It)

**Format**

```
#include "sys/unistd.h"
int rresvport(int *port)
```

---

**Parameters**

int *	port	Pointer to the buffer where the port number is stored
-------	------	---

---

**Returned value**

int	A socket file descriptor bound to a port. This function returns -1 if no socket is present.
-----	---

---

**errno**

Not specified
---------------

---

#### 4.2.24 *getifaddrs* (Retrieve Interface Addresses)

##### Format

```
#include "sys/types.h"
int getifaddrs(struct ifaddrs **ifap)
```

##### Parameters

struct ifaddrs**	ifap	The address of the first item in the list of network interfaces
------------------	------	---

##### Returned value

int	Result of processing. This function returns 0 on success and -1 if an error occurred.
-----	---

##### errno

ENOMEM	Failure to acquire the area where the information about the interfaces is stored
--------	--

- This function acquires the information about the interfaces for the devices (CFG\_DEV\_MAX) set in the application.
- On success, this function stores the following values in the argument ifap.
  - (\*ifap)->ifa\_next : a pointer to the next structure in the list, or null if this is the last item in the list
  - (\*ifap)->name : a pointer to the interface name
  - (\*ifap)->ifa\_flags : the device number
  - (\*ifap)->ifa\_addr : a pointer to the sockaddr structure which contains the IP address of the interface
  - (\*ifap)->ifa\_netmask : a pointer to the sockaddr structure which contains the subnet mask
  - (\*ifap)->ifa\_ifu and (\*ifap)->ifa\_data : not used in this function
- The data returned by getifaddrs() is dynamically allocated and should be freed by using freeifaddrs() after the function succeeds.

#### 4.2.25 *freeifaddrs* (Free List of Interface Information)

##### Format

```
#include "sys/unistd.h"
void freeifaddrs(struct ifaddrs *ifa)
```

---

##### Parameters

struct ifaddrs*	ifa	The address of the first item in the list of network interfaces
-----------------	-----	---

---

##### Returned value

void

---

##### errno

Not specified

---

- This function frees the list of interface information acquired by `getifaddrs()`.



## 5. Socket Options

### 5.1 List of Options

Table 5.1, List of Options is the options acquired or set by using the functions `setsockopt()` and `getsockopt()`. If a value other than those listed below is specified, the function returns `-1`.

In the list, “GET” represents operations to which `getsockopt()` is applicable and “SET” represents operations to which `setsockopt()` is applicable.

Table 5.1 List of Options

Option Name	Type	Description
SOL_SOCKET Level		
SO_ACCEPTCONN	int	Retrieve the state of a TCP socket, whether it is in listening mode or not. Only GET is applicable.
SO_BROADCAST	int	Configure a socket for receiving UDP broadcast data. Both GET and SET are applicable.
SO_DOMAIN	int	Acquire the socket domain. Only GET is applicable.
SO_ERROR	int	Acquire a socket error. Only GET is applicable.
SO_KEEPALIVE	int	Enable sending of keepalive packets by the TCP socket. Only SET is applicable.
SO_RCVBUF	int	Make settings for the reception buffer. This is the number of bytes in reception windows for TCP and the number of received packets (queue size) for the UDP. Both GET and SET are applicable.
SO_RCVBUFFORCE	int	Same as SO_RCVBUF
SO_RCVTIMEO	timeval	Specify the timeout value for a receiving socket. Both GET and SET are applicable.
SO_SNDTIMEO	timeval	Specifies the timeout value for a sending socket. Both GET and SET are applicable.
SO_TYPE	int	Retrieves the socket type. Only GET is applicable.
IPPROTO_IP Level		
IP_ADD_MEMBERSHIP	ip_mreqn	Joins the multicast groups specified, applicable to UDP sockets only. Only SET is applicable.
IP_DROP_MEMBERSHIP	ip_mreqn	Drops membership of a multicast group. Only SET is applicable.
IP_MTU	int	Retrieve the path MTU. Only GET is applicable.
IP_MULTICAST_TTL	int	Set the TTL (time-to-live) for transmitted multicast packets. Both GET and SET are applicable.
IP_TOS	int	Set the TOS (type of service) for transmitted IP packets. Both GET and SET are applicable.
IP_TTL	int	Set the TTL for transmitted IP packets. Both GET and SET are applicable.
IPPROTO_TCP Level		
TCP_KEEPCNT	int	Specifies the number of keepalive probes for TCP sockets. Only SET is applicable.
TCP_KEEPIIDLE	int	Specifies the interval of inactivity that causes the TCP to generate a keepalive transmission for an application that requests them. Only SET is applicable.
TCP_KEEPINTVL	int	Specifies the interval between keepalive probes for TCP sockets. Only SET is applicable.
TCP_MAXSEG	int	Specifies the MSS (maximum segment size) value for TCP packets. Both GET and SET are applicable.

## 6. Capabilities

### 6.1 Non-Blocking Mode

The `ioctl()` function sets the API call for a socket in non-blocking mode (or blocking mode). All API functions are set to blocking mode as the initial value. There are some cases where an API in non-blocking mode sets `EAGAIN` as `errno` and returns `-1`. The APIs which operated in non-blocking mode and the conditions for returning `EAGAIN` as `errno`, and the expected behaviors of the application are listed in Table 6.1, Non-Blocking APIs.

Note that setting the timeout option for a socket is not effective for the APIs which behave in non-blocking mode. Furthermore, in  $\mu$ Net3/BSD, even if an API function is set to non-blocking mode, it may need to wait for the task to wake up after being called due to the specification for inter-task transfer.

Table 6.1 Non-Blocking APIs

API	Condition	Application Behavior
<code>connect</code>	If the target is a TCP socket, the returned value is always <code>-1</code> , and the error code otherwise is <code>EAGAIN</code> .	Even after <code>-1</code> is returned, the TCP socket keeps sending SYN packets for the specified time while waiting for SYN and ACK packets from the remote party. The socket is monitored by the <code>select</code> function with the parameter <code>writelfds</code> , for readiness for writing on reception of SYN and ACK. Once the socket becomes ready for writing, further execution of the <code>connect</code> function is not needed.
<code>accept</code>	If there is no connection attempted to the listen socket, the returned value is <code>-1</code> and the error code is <code>EAGAIN</code> .	The socket is monitored by the <code>select</code> function with the parameter <code>readfds</code> , for readiness for reading on reception of SYN. Once the socket becomes ready for reading, the <code>accept</code> function is executed again.
<code>send</code> <code>sendto</code>	When the send buffer is full in the transfer with the TCP sockets and when a transmission is in progress in the transfer with the UDP sockets, the error code is <code>EAGAIN</code> .	<code>EAGAIN</code> for the functions <code>send</code> and <code>sendto</code> means that packet transmission failed (and will not be transmitted again) due to conditions of sockets.
<code>recv</code> <code>recvfrom</code>	When no packet has been received, the error code is <code>EAGAIN</code> .	The socket is monitored by the <code>select</code> function with the parameter <code>readfds</code> , for readiness for reading on reception of packets from the remote party. Once the socket becomes ready for reading, the <code>recv</code> function is executed again.

## 6.2 Loopback

When local loopback addresses (127.0.0.1 to 127.255.255.254) are specified as destination for transmission, the transmitted packets are conveyed to the network interface of the local device.

In μNet3/BSD, the loopback addresses are not assigned to any specific device interface and are not regarded as send-only addresses. Therefore, they cannot be used in the bind() operation.

## 6.3 Error Processing

The symbol `errno` is the only global variable used in the μNet3/BSD API. Its value is updated on the occurrence of errors during the execution of API functions. When the user executes API functions from multiple tasks, we recommend acquiring the last `errno` by using `get_errno()`, to maintain consistency between `errno` values and the errors.

### Format

```
#include "sys/errno.h"
int get_errno(void)
```

### Parameters

void

### Returned value

int                                      The `errno` of the last error to have occurred during the API function calls by a given task

### `errno`

Not specified

- The `errno` for each task will be stored in the global variable `UW_tsk_errno[]`, provided in the application. The array should have the same number of elements as the maximum number of tasks.

## 6.4 List of errno

The values defined for errno may vary according to the compiler.

[Definition pattern 1]

Applicable compilers:

- RealView Developer Suite from ARM
- Embedded Workbench (EWARM) from IAR
- Code Composer Studio from TI
- GNU C Compiler from GNU

errno	Value	Description
ENXIO	6	No interface found
EBADF	9	Invalid socket file descriptor
ENOMEM	12	Not enough memory
EFAULT	14	Error in a parameter
ENODEV	19	Critical (or unknown) error in the system
EINVAL	22	Invalid parameter value
EPIPE	32	Invalid socket object
EAGAIN	35	Connection is blocked.
EALREADY	37	The operation is already in progress.
EDESTADDRREQ	39	Destination address required
EPROTONOSUPPORT	43	Function not supported
EAFNOSUPPORT	47	Address family not supported by protocol
EADDRINUSE	48	Address already in use
EADDRNOTAVAIL	49	Cannot assign requested address
EISCONN	56	The socket is already connected.
ENOTCONN	57	The socket is not connected.
ETIMEDOUT	60	Connection attempt timed out
ECONNREFUSED	61	Connection is refused by server.

[Definition pattern 2]

Applicable compiler:

- CubeSuite+ \* from Renesas Electronics

errno	Value	Description
ENXIO	*	No interface found
EBADF	*	Invalid socket file descriptor
ENOMEM	*	Not enough memory
EFAULT	*	Error in a parameter
ENODEV	*	Critical (or unknown) error in the system
EINVAL	*	Invalid parameter value
EPIPE	*	Invalid socket object
EAGAIN	*	Connection is blocked.
EALREADY	0x1025	The operation is already in progress.
EDESTADDRREQ	0x1027	Destination address required
EPROTONOSUPPORT	0x102B	Function not supported
EAFNOSUPPORT	0x102F	Address family not supported by protocol
EADDRINUSE	0x1030	Address already in use
EADDRNOTAVAIL	0x1031	Cannot assign requested address
EISCONN	0x1038	The socket is already connected.
ENOTCONN	0x1039	The socket is not connected.
ETIMEDOUT	*	Connection attempt timed out
ECONNREFUSED	0x103D	Connection is refused by server.

- Restrictions

- Do not use the name “errno” as the name of a variable where error codes are stored.  
The name “errno” is used as the variable defining errors in the CubeSuite+ standard library. With the μNet3/BSD API, use *unet\_errno* instead.
- Part of the values defined for errno are the same as those used in the compiler (as indicated with \* in the above list).  
If the same name is defined for an errno value in the μNet3/BSD API and the CubeSuite+, the one in the compiler is used.

## 7. Implementing BSD Application

### 7.1 Source Code

An application which uses the μNet3/BSD API must be combined in projects with four files of source code from the Network/bsd/ folder (see Section 3.3, Source Files).

Also, link the version of μNet3/BSD which is prepared for BSD (uNet3BSDxxxx.lib) as the library.

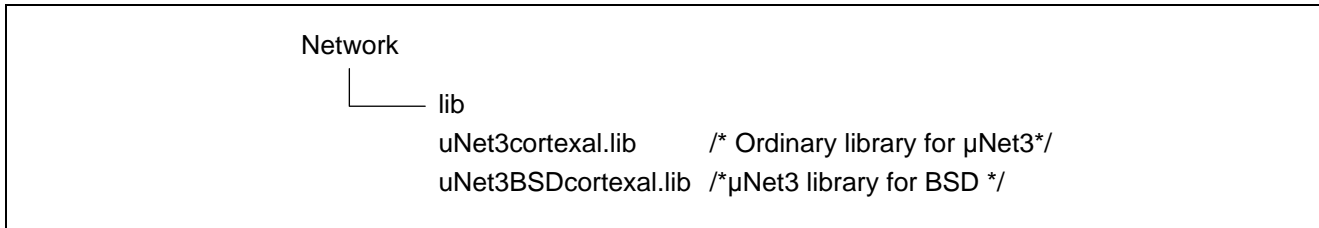


Figure 7.1 μNet3 Library

### 7.2 Include Path

An application which uses the μNet3/BSD API requires additional settings for “include” paths. The header file is found in the Network/bsd/unet3\_posix folder with the POSIX-compatible files.

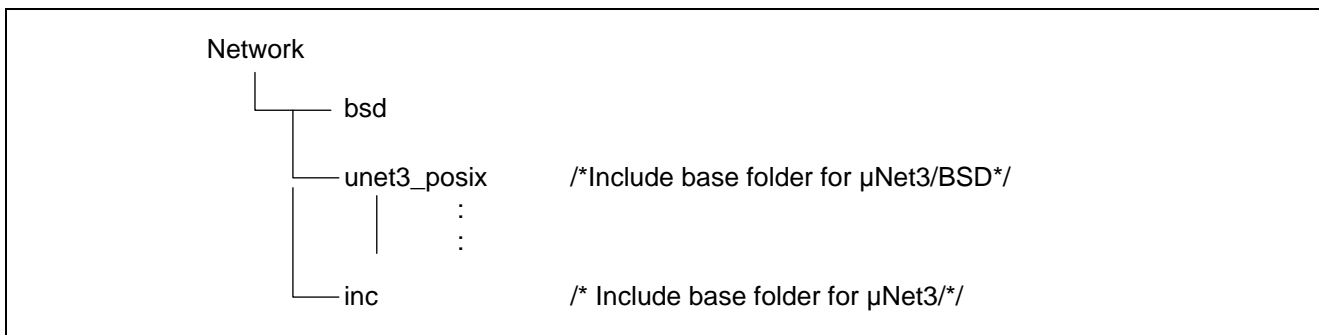


Figure 7.2 "Include" Paths for μNet3

### 7.3 Configuration

In the  $\mu$ Net3/BSD API, the maximum number of sockets to be used in an application and tasks to be executed should be defined in the below macro in `unet3_cfg.h` in advance.

#### Maximum number of sockets

```
#define    BSD_SOCKET_MAX
```

The maximum number of sockets, regardless of the protocol, shows the number of sockets that an application can create at the same time (including backlog from listening). This macro definition is used for defining the number of entries in the management table for BSD sockets and `fd_set` type settings, which will be described later. This value must be same as the maximum number of the sockets used by  $\mu$ Net3 (`CFG_NET_SOC_MAX`).

#### Number of application tasks

```
#define    NUM_OF_TASK_ERRNO
```

The number of application tasks shows the number of tasks that can be created in the kernel. This macro definition is used for the number of entries in the management table for error codes, which will be described later. Set the number of tasks that can be created, regardless of whether you are using  $\mu$ Net3/BSD.

## 7.4 Defining Resources

Applications which use the μNet3/BSD API should provide resources required for operating the program, which are, the tables for managing information of the μNet3/BSD as listed below.

### BSD socket management table

```
T_UNET3_BSD_SOC    gNET_BSD_SOC[BSD_SOCKET_MAX];
```

This table defines a global variable as the number of elements BSD\_SOCKET\_MAX in the T\_UNET3\_BSD\_SOC array.

### Error number management table

```
UW_tsk_errno[NUM_OF_TASK_ERRNO];
```

This table defines a global variable as the number of elements NUM\_OF\_TASK\_ERRNO in the array of UW.



## 7.5 Kernel Objects

The Kernel objects used in the μNet3/BSD are shown below.

Resource Name	Usage	ID
Task	BSD wrapper task	ID TSK_BSD_API
	Loopback device task	ID_LO_IF_TSK
Mailbox	Communication between BSD wrapper tasks	ID MBX_BSD_REQ
	Communication between loopback device tasks	ID_LO_IF_MBX
Memory pool	Message buffer	ID MPF_BSD_MSG

## 7.6 Initialization

When an application uses the socket API functions of the μNet3/BSD API, the module must be initialized in advance by calling the `unet3_bsd_init()` function. This operation should be performed after successful initialization of μNet3 and the device driver.

### Format

```
#include "sys/socket.h"
ER unet3_bsd_init(void)
```

### Parameters

void

### Returned value

ER                      Result of processing. This function returns E\_OK on success and the error code if an error occurred.

### Error code

E\_SYS                    Initialization of the kernel object failed

## 8. Appendix

### 8.1 Supported Compilers

The  $\mu$ Net3/BSD guarantees operation in the following compilers.

- RealView Developer Suite from ARM
- Embedded Workbench (EWARM) from IAR
- Code Composer Studio from TI
- GNU C Compiler from GNU
- CubeSuite+ from Renesas Electronics

Note: Restrictions are given depending on the compiler.

### 8.2 Sample Application

Sample applications using the  $\mu$ Net3/BSD API are included in the Sample folder. These sample programs are also available in the POSIX environment (Linux).

- API Console (sample\_sockcmd.c)

The user can run the API functions through the command prompt (with a UART connection) by input of the socket API function and required parameters. For details, refer to the Readme\_command.txt.

### 8.3 Restrictions on Compilers

Restrictions are given to some compilers when they are used with  $\mu$ Net3/BSD.

- CubeSuite+ from Renesas Electronics
  - Do not use the name “errno” as the name of a variable where error codes are stored.  
The name “errno” is used as the variable defining errors in the CubeSuite+ standard library. With the  $\mu$ Net3/BSD API, use *unet\_errno* instead.

---

REVISION HISTORY	R-IN32 Series User's Manual ( $\mu$ Net3/BSD edition)
------------------	---

Rev.	Date	Description	
		Page	Summary
1.00	Sep 5, 2016	-	First edition issued

[MEMO]

R-IN32 Series User's Manual  
( $\mu$ Net3/BSD edition)



Renesas Electronics Corporation

**SALES OFFICES**

<http://www.renesas.com>

Refer to "http://www.renesas.com/" for the latest and detailed information.

**Renesas Electronics America Inc.**

2801 Scott Boulevard Santa Clara, CA 95050-2549, U.S.A.  
Tel: +1-408-588-6000, Fax: +1-408-588-6130

**Renesas Electronics Canada Limited**

9251 Yonge Street, Suite 8309 Richmond Hill, Ontario Canada L4C 9T3  
Tel: +1-905-237-2004

**Renesas Electronics Europe Limited**

Dukes Meadow, Millboard Road, Bourne End, Buckinghamshire, SL8 5FH, U.K  
Tel: +44-1628-585-100, Fax: +44-1628-585-900

**Renesas Electronics Europe GmbH**

Arcadiastrasse 10, 40472 Düsseldorf, Germany  
Tel: +49-211-6503-0, Fax: +49-211-6503-1327

**Renesas Electronics (China) Co., Ltd.**

Room 1709, Quantum Plaza, No.27 ZhiChunLu Haidian District, Beijing 100191, P.R.China  
Tel: +86-10-8235-1155, Fax: +86-10-8235-7679

**Renesas Electronics (Shanghai) Co., Ltd.**

Unit 301, Tower A, Central Towers, 555 Langao Road, Putuo District, Shanghai, P. R. China 200333  
Tel: +86-21-2226-0888, Fax: +86-21-2226-0999

**Renesas Electronics Hong Kong Limited**

Unit 1601-1611, 16/F., Tower 2, Grand Century Place, 193 Prince Edward Road West, Mongkok, Kowloon, Hong Kong  
Tel: +852-2265-6688, Fax: +852 2886-9022

**Renesas Electronics Taiwan Co., Ltd.**

13F, No. 363, Fu Shing North Road, Taipei 10543, Taiwan  
Tel: +886-2-8175-9600, Fax: +886 2-8175-9670

**Renesas Electronics Singapore Pte. Ltd.**

80 Bendemeer Road, Unit #06-02 Hyflux Innovation Centre, Singapore 339949  
Tel: +65-6213-0200, Fax: +65-6213-0300

**Renesas Electronics Malaysia Sdn.Bhd.**

Unit 1207, Block B, Menara Amcorp, Amcorp Trade Centre, No. 18, Jln Persiaran Barat, 46050 Petaling Jaya, Selangor Darul Ehsan, Malaysia  
Tel: +60-3-7955-9390, Fax: +60-3-7955-9510

**Renesas Electronics India Pvt. Ltd.**

No.777C, 100 Feet Road, HALII Stage, Indiranagar, Bangalore, India  
Tel: +91-80-67208700, Fax: +91-80-67208777

**Renesas Electronics Korea Co., Ltd.**

12F., 234 Teheran-ro, Gangnam-Gu, Seoul, 135-080, Korea  
Tel: +82-2-558-3737, Fax: +82-2-558-5141