Network Enabler SDK 2 API Reference

Eighth Edition, June 2008

www.moxa.com/product

Network Enabler SDK 2 API Reference

Technical Support Contact Information
www.moxa.com/support

Moxa Americas:
Toll-free: 1-888-669-2872
Tel: +1-714-528-6777
Fax: +1-714-528-6778

Moxa China (Shanghai office):
Toll-free: 800-820-5036
Tel: +86-21-5258-9955
Fax: +86-10-6872-3958

Moxa Europe:
Tel: +49-89-3 70 03 99-0
Fax: +49-89-3 70 03 99-99

Moxa Asia-Pacific:
Tel: +886-2-8919-1230
Fax: +886-2-8919-1231

Table of Contents

1 Overview

The **Network Enabler SDK 2 API Reference** is your complete guide to the Application Program Interface (API) function calls and linking libraries that are available in Moxa's Network Enabler Software Development Kit (SDK). You may also refer to the companion guide, the **Network Enabler SDK 2 Programmer's Guide**.

The following topic is covered in this chapter:

- ❑ **Sample Layout**

Sample Layout

The SDK API functions are displayed in the format shown below.

function name	*brief function description*	*function type*

Syntax

`#include` **<***header file name***>**
function call syntax

Arguments

variable names	*brief description of variables*

Description

detailed function description

Return Value

return code #1	*description of return code*
return code #2	*description of return code*

The function **`sio_oqueue`** is shown here as an example from the SDK API Serial I/O library. This function reports the amount of data that is waiting to be transmitted out through the serial port.

`sio_oqueue`	get the length of data not yet sent out in both the system's output buffer and the driver's output buffer	**Port Status**

Syntax

```
#include  <sdksio.h>
long  sio_oqueue ( int port );
```

Arguments

`port`	async serial port number

Description

get the length of data not yet sent out in both the system's output buffer and the driver's output buffer

Return Value

`>= 0`	length of data (bytes) still remaining driver's output buffer
`SIO_BADPORT`	port number is invalid

2
API List

The Network Enabler SDK includes some programming utilities for use with the NE-4100-P and user-developed applications. Several detailed sample programs are also provided. You may refer to the companion guide, Network Enabler SDK Programmer's Guide, for additional information about using the utilities.

In order to make the SDK library easier to use, function calls are divided into categories as shown below. The categories are intended to assist programmers in finding the correct function call for their application.

This chapter lists every function call with a brief description. Detailed information on each function call can be found in Chapter 3. The following topics are covered in this chapter:

- ❑ **Overview**
- ❑ **Serial I/O API**
- ❑ **BSD Socket API**
- ❑ **Simplified Socket API**
- ❑ **System Control API**
- ❑ **Flash ROM Access API**
- ❑ **Debug API**
- ❑ **DIO API**
- ❑ **Thread Control API**
- ❑ **Time Server API**

Overview

This chapter presents a broad overview of every function call, grouped by category and function type. Please refer to Chapter 3 for detailed descriptions of each function.

For each library category, there is a specific header file that needs to be included in the source code when calling functions within that category. Please refer to the example source code for details of how to include a header file.

Serial I/O API

The header file **`sdksio.h`** must be included in your source code when calling serial I/O functions.

Port Control

Port control functions are used to open serial ports, set communication parameters, and control signal lines.

Function Name	**Description**
`sio_baud`	set baud rate using the actual speed value
`sio_close`	stop receiving/transmitting data
`sio_DTR`	set DTR state
`sio_flowctrl`	set port hardware or software flow control
`sio_flush`	flush input or output buffer
`sio_ioctl`	set port baud rate, parity, etc.
`sio_lctrl`	set DTR and RTS states
`sio_open`	start receiving/transmitting data
`sio_RTS`	set RTS state

Data Input

Data input functions are used to read data from the COM port.

Function Name	**Description**
`sio_AbortRead`	abort when reading a block of data for **`sio_read()`**
`sio_getch`	read one character at a time from driver's input buffer
`sio_GetReadTimeouts`	get timeouts for **`sio_read()`**
`sio_linput`	read a block of data ending with a termination character
`sio_read`	read a block of data from the driver's input buffer
`sio_SetReadTimeouts`	set timeouts for **`sio_read`**

Data Output

Data output functions are used to write data to the serial port.

Function Name	**Description**
`sio_AbortWrite`	abort when writing a block of data for **`sio_write()`**
`sio_GetWriteTimeouts`	get timeouts for **`sio_write()`**
`sio_putch`	write one character at a time to driver's output buffer
`sio_SetWriteTimeouts`	set timeouts for **`sio_write()`**
`sio_write`	write a block of data (usually only a partial block) to output buffer

Port Status

Port status functions are used to query the communication status from the serial port.

Function Name	Description
`sio_data_status`	check if error occurred when receiving data
`sio_getbaud`	get baud rate setting
`sio_getflow`	get hardware and software flow control settings
`sio_getmode`	get settings for parity, data bits, etc
`sio_iqueue`	get length of data accumulated in driver's input buffer
`sio_lstatus`	get line status
`sio_ofree`	get amount of free space in driver’s output buffer
`sio_oqueue`	get length of data still held in driver's output buffer
`sio_Tx_hold`	check why data could not be transmitted

Event Control

Event control functions are used to set the communication event service routines for the serial port.

Function Name	Description
`sio_break_irq`	set event service routine for when break signal is received
`sio_cnt_irq`	set event service routine for when a certain amount of data is received
`sio_modem_irq`	set event service routine for when line status is changed
`sio_term_irq`	set event service routine for when termination character is received
`sio_Tx_empty_irq`	set event service routine for when transmit buffer is empty

Miscellaneous

Miscellaneous functions are special COM port functions.

Function Name	Description
`sio_ActXoff`	cause transmission to act as if an XOFF character has been received
`sio_ActXon`	cause transmission to act as if an XON character has been received
`sio_break`	send out BREAK signal
`sio_break_ex`	send out BREAK signal

BSD Socket API

The header file **`sdksock.h`** must be included in your source code when calling BSD socket functions.

Socket Control

Socket control functions are used to open TCP sockets, and set and retrieve communication parameters.

Function Name	**Description**
`accept`	acknowledge an incoming connection and associate it with an immediately created socket; return original socket to listening state
`bind`	assign a local name to an unnamed socket
`closesocket`	remove a socket from the per-process object reference table; only blocks if **`SO_LINGER`** is set
`connect`	initiate a connection on the specified socket
`getsockopt`	retrieve options associated with the specified socket
`ioctlsocket`	provide control of sockets
`listen`	listen for incoming connections on a specified socket
`setsockopt`	store options associated with the specified socket
`shutdown`	shut down part of a full-duplex connection
`socket`	create an endpoint for communication and return a socket

Data Input/Output

Data input and output functions are used to read and write data from the socket.

Function Name	**Description**
`recv`	receive data from a connected socket
`recvfrom`	receive data from either a connected or unconnected socket
`select`	perform synchronous I/O multiplexing
`send`	send data to a connected socket
`sendto`	send data to either a connected or unconnected socket

Inquiry

Inquiry functions are used to query the communication status from the socket.

Function Name	**Description**
`gethostbyname`	retrieve name(s) and address corresponding to a host name
`gethostname`	retrieve name of the local host
`getpeername`	retrieve name of the peer connected to the specified socket
`getsockname`	retrieve current name for the specified socket

Miscellaneous

Miscellaneous functions are special socket functions.

Function Name	Description
`htonl`	convert an unsigned long from host to network byte order
`htons`	convert an unsigned short from host to network byte order
`inet_addr`	convert a string containing a dotted address into a long integer
`inet_ntoa`	convert a network address into a string in dotted format
`ntohl`	convert an unsigned long from network to host byte order
`ntohs`	convert an unsigned short from network to host byte order

Simplified Socket API

The header files **`sdknet.h`** and **`socksys.h`** must be included in your source code when calling simplified socket functions.

Socket Control

Socket control functions are used to open TCP/UDP sockets, and set and retrieve communication parameters.

Function Name	Description
`tcp_close`	close local TCP port
`tcp_connect`	connect to specific host IP and port
`tcp_connect_nowait`	connect to specific host IP and port no wait
`tcp_listen`	place socket in a state where it is listening for an incoming connection
`tcp_listen_nowait`	place socket in a state where it is listening for an incoming connection no wait
`tcp_listento`	listen for a specific incoming connection
`tcp_listento_nowait`	listen for a specific incoming connection no wait
`tcp_open`	open local TCP port
`udp_close`	close local UDP port
`udp_open`	open local UDP port

Data Input/Output

Data input and output functions are used to read and write data from the socket.

Function Name	Description
`tcp_recv`	receive data from a connected socket
`tcp_send`	send data out through a connected socket
`udp_recv`	receive data from a specific source address
`udp_send`	send data to a specific destination

Socket Inquiry

Socket inquiry functions are used to query the communication status of the socket.

Function Name	Description
`tcp_get_remote`	get connected host IP and port
`tcp_iqueue`	get length of data accumulated in TCP driver’s input buffer
`tcp_ofree`	get amount of free space in TCP driver’s input buffer

Function Name	Description
`tcp_state`	get TCP state
`udp_iqueue`	get length of data accumulated in UDP driver’s input buffer
`udp_ofree`	get amount of free space in UDP driver’s input buffer

Port Inquiry

Port inquiry functions are used to query current Ethernet port status and parameters.

Function Name	Description
`net_get_gateway`	get local default gateway
`net_get_IP`	get local IP address
`net_get_MAC_address`	get MAC address
`net_get_netmask`	get local subnet mask

System Control API

The header file **`sdksys.h`** must be included in your source code when calling system control functions.

Function Name	Description
`sys_calloc`	allocate an array with a specific amount of memory
`sys_clock_ms`	read the server’s time (milliseconds) measured from power-up
`sys_clock_s`	read the server’s time (seconds) measured from power-up
`sys_exit`	exit application
`sys_free`	free up a specified amount of memory
`sys_get_info`	get server’s general information
`sys_get_LastErrno`	get last error number related to a socket
`sys_get_SerialType`	get current async port interface signal type
`sys_getFreeMemSize`	get the amount of free memory space
`sys_GetServersIp`	get DNS server’s IP address
`sys_malloc`	allocate a specific amount of memory
`sys_realloc`	re-allocate a specific amount of memory
`sys_restart_system`	restart system
`sys_restart_UserAP`	restart user AP
`sys_Set_RegisterID`	set AP ID
`sys_set_SerialType`	set async port interface signal type
`sys_sleep_ms`	get task sleep time (milliseconds)
`sys_timeout`	set timeout event service routine
`sysc_GetDebug`	get debug output setting
`sysc_GetGateway`	get specified network interface gateway
`sysc_GetIP`	get specified network interface IP address
`sysc_GetIPConfig`	get IP configuration settings
`sysc_GetIPLocating`	get IP Location setting
`sysc_GetName`	get server name
`sysc_GetNetmask`	get specified network interface netmask
`sysc_GetPassword`	get server password
`sysc_GetSerialFIFO`	get serial port FIFO settings

Function Name	Description
sysc_GetSerialInterface	get serial port interface
sysc_GetSerialIoctl	get serial port parameters
sysc_SaveAndRestart	save new settings and restart NE-4100-P
sysc_SetDebug	set debug output setting
sysc_SetGateway	set gateway address
sysc_SetIP	set the specified network interface IP address
sysc_SetIPConfig	define how to get IP address, netmask and gateway
sysc_SetIPLocating	set IP Location function
sysc_SetName	set server name
sysc_SetNetmask	set netmask
sysc_SetPassword	set password
sysc_SetSerialFIFO	set serial port FIFO settings
sysc_SetSerialInterface	set serial port interface
sysc_SetSerialIoctl	set serial port parameters
sysc_SetToDefault	set to default values

Flash ROM Access API

The header file **sdkflash.h** must be included in your source code when calling flash ROM access functions.

Function Name	Description
flash_erase	erase flash ROM
flash_length	get length of data in flash ROM
flash_read	read data from flash ROM
flash_write	write data to flash ROM
sys_FlashErase	erase flash ROM
sys_FlashLength	get length of data in flash ROM
sys_FlashRead	read data to flash ROM
sys_FlashWrite	write data to flash ROM

Debug API

The header file **sdkdbg.h** must be included in your source code when calling debug functions.

Function Name	Description
dbg_printf	print formatted output to debug output stream
dbg_put_block	print out a block of data for debugging
dbg_put_ch	print out a character for debugging
dbg_put_doubleword	print out a 4-byte unsigned long value for debugging
dbg_put_doubleword_hex	print out a 4-byte unsigned long value with HEX format for debugging
dbg_put_IP	print out an IP address in a.b.c.d format for debugging
dbg_put_string	print out a string for debugging
dbg_put_word	print out a 2-byte unsigned integer value for debugging
dbg_put_word_hex	print out a 2-byte unsigned integer value with HEX format for debugging

DIO API

The header file **`sdkdio.h`** must be included in your source code when calling DIO functions.

Function Name	Description
`DIO_ControlSingleIO`	set output channel state to high or low
`DIO_GetSingleIO`	get I/O channel's mode (input or output)
`DIO_GetSingleIOStatus`	get output channel's state (high or low)
`DIO_SetSingleIO`	set I/O channel's mode to input or output

Thread Control API

The header file **`sdktask.h`** must be included in your source code when calling thread control functions.

Function Name	Description
`sys_ThreadClose`	close a thread
`sys_ThreadCreate`	create a thread
`sys_ThreadResume`	resume a thread
`sys_ThreadState`	get a thread state
`sys_ThreadSuspend`	suspend a thread

Time Server API

The header files **`sdkconf.h`** and **`sdksys.h`** must be included in your source code when calling time server functions.

Function Name	Description
`sys_GetLocalTime`	get local time
`sys_SetLocalTime`	set local time
`sysc_getTimeServer`	get time server
`sysc_getTimeZone`	get time zone
`sysc_getTZoneIndex`	get time zone index
`sysc_setTimeServer`	set time server
`sysc_setTimeZone`	set time zone
`sysc_setTZoneIndex`	set time zone index

ATTENTION

The NE-4100-P uses a software timer to simulate a real time clock. NTP (Network Time Protocol) is used to synchronize the date and time of the internal clock with time server. If time information cannot be obtained due to network trouble, the system time will be set to Jan.1, 2000.

3
API Reference

The following topics are covered in this chapter:

- ❑ **Serial I/O Library Reference**
- ❑ **BSD Socket Library Reference**
- ❑ **Simplified Socket Library Reference**
- ❑ **System Control Library Reference**
- ❑ **Flash ROM Access Library Reference**
- ❑ **Debug Library Reference**
- ❑ **DIO Library Reference**
- ❑ **Thread Control Library Reference**
- ❑ **Time Server Library Reference**
- ❑ **Time Zone Offsets Index**

Serial I/O Library Reference

sio_AbortRead	abort when blocked from reading a block of data for **sio_read()** and **sio_getch()**	**Data Input**

Syntax
```
#include  <sdksio.h>
int  sio_AbortRead ( int port );
```

Arguments

port	async serial port number

Description

abort when blocked from reading a block of data for **sio_read()** and **sio_getch(**); calling this function will cause **sio_read()** to return immediately with return code of length of data read

Return Value

SIO_OK	OK
SIO_BADPORT	port number is invalid

sio_AbortWrite	abort when blocked from writing a block of data for **sio_write()** and **sio_putch()**	**Data Output**

Syntax
```
#include  <sdksio.h>
int  sio_AbortWrite ( int port );
```

Arguments

port	async serial port number

Description

abort when blocked from writing a block of data for **sio_write()** or **sio_putch()**; calling this function will cause **sio_write()** to return immediately with return code of **SIO_ABORT_WRITE**

Return Value

SIO_OK	OK
SIO_BADPORT	port number is invalid

sio_ActXoff	make transmission act as if an XOFF character has been received	**Misc.**

Syntax
```
#include  <sdksio.h>
int  sio_ActXoff ( int port );
```

Arguments

port	async serial port number

Description

this function causes transmission to act as if an XOFF character has been received

Return Value

SIO_OK	OK
SIO_BADPORT	port was not open in advance

sio_ActXon	make transmission act as if an XON character has been received	**Misc.**

Syntax
```
#include  <sdksio.h>
int  sio_ActXon ( int port );
```
Arguments

port	async serial port number

Description

this function causes transmission to act as if an XON character has been received

Return Value

SIO_OK	OK
SIO_BADPORT	port number is invalid

sio_baud	set baud rate using the actual speed value	**Port Control**

Syntax
```
#include  <sdksio.h>
int  sio_baud ( int port, long speed );
```
Arguments

port	async serial port number
speed	true baud rate: e.g., 200, 1200, 9600, or 19200

Description

set baud rate using the actual speed value

Return Value

SIO_OK	OK
SIO_BADPORT	port number is invalid

sio_break	send out a BREAK signal	**Misc.**

Syntax
```
#include  <sdksio.h>
int  sio_break ( int port, int time );
```
Arguments

port	async serial port number
time	break time in tics (1/18.2 second)

Description

this function will block transmission until the time has expired

Return Value

SIO_OK	OK
SIO_BADPORT	port number is invalid
SIO_BADPARM	bad parameter
SIO_NOT_OPEN	port was not open in advance

sio_break_ex	send out a BREAK signal	**Misc.**

Syntax

```
#include  <sdksio.h>
int  sio_break_ex ( int port, int ms );
```

Arguments

port	async serial port number
ms	break time (milliseconds)

Description

sends out a break signal; will block transmission until time has expired; is the same as **sio_break()**, except that the time unit is measured in milliseconds

Return Value

SIO_OK	OK
SIO_BADPORT	port number is invalid
SIO_BADPARM	bad parameter
SIO_NOT_OPEN	port was not open in advance

sio_break_irq	set an event service routine to be called when a BREAK signal is received	**Event Control**

Syntax

```
#include  <sdksio.h>
int  sio_break_irq ( int port, void (*func) (int port) );
```

Arguments

port	async serial port number
func	event service routine entry; if **func** is NULL, this routine will be disabled

Description

set an event service routine to be called when a BREAK signal is received; when a BREAK signal is encountered, the system will call the event service routine

Return Value

SIO_OK	OK
SIO_BADPORT	port number is invalid
SIO_NOT_OPEN	port was not open in advance

sio_close	disable serial port for transmitting or receiving data	**Port Control**

Syntax

```
#include  <sdksio.h>
int  sio_close ( int port );
```

Arguments

port	async serial port number

Description

disable a serial port so that it cannot receive or transmit data

Return Value

SIO_OK	OK
SIO_BADPORT	port number is invalid
SIO_NOT_OPEN	port was not open in advance

`sio_cnt_irq`	set an event service routine to be called when a certain amount of data has been received	**Event Control**

Syntax

```
#include  <sdksio.h>
int  sio_cnt_irq ( int port, void (*func) (int port), int count );
```

Arguments

`port`	async serial port number
`func`	event service routine entry; if **`func`** is NULL, this routine will be disabled
`count`	data count

Description

set an event service routine to be called when a certain amount of data has been received; when there are **`count`** bytes of data received in the input buffer, the system will call the **`func`** service routine

Return Value

`SIO_OK`	OK
`SIO_BADPORT`	port number is invalid
`SIO_NOT_OPEN`	port was not open in advance

`sio_data_status`	check if error occurred when receiving data	**Port Status**

Syntax

```
#include  <sdksio.h>
int  sio_data_status ( int port );
```

Arguments

`port`	async serial port number

Description

check if an error occurred when receiving data

Return Value

`0`	no error
`>0`	bit 0 on: parity error bit 1 on: framing error bit 2 on: overrun error bit 3 on: overflow error
`SIO_BADPORT`	port number is invalid

sio_DTR	set DTR state of a port	**Port Control**

Syntax

```
#include  <sdksio.h>
int  sio_DTR ( int port, int mode );
```

Arguments

port	async serial port number
mode	0: turn DTR off 1: turn DTR on

Description

set the DTR state of a port

Return Value

SIO_OK	OK
SIO_BADPORT	port number is invalid
SIO_BADPARM	bad parameter
SIO_NOT_OPEN	port was not open in advance

sio_flowctrl	set hardware and/or software flow control	**Port Control**

Syntax

```
#include  <sdksio.h>
int  sio_flowctrl ( int port, int mode );
```

Arguments

port	async serial port number
mode	bit 0: CTS flow control bit 1: RTS flow control bit 2: Tx XON/XOFF flow control bit 3: Rx XON/XOFF flow control (0 = OFF, 1 = ON)

Description

set the hardware and/or software flow control

Return Value

SIO_OK	OK
SIO_BADPORT	port number is invalid
SIO_BADPARM	bad parameter
SIO_NOT_OPEN	port was not open in advance

sio_flush	flush the driver's input/output buffer	**Port Control**

Syntax

```
#include  <sdksio.h>
int  sio_flush ( int port, int func );
```

Arguments

port	async serial port number
func	flush action 0: flush input buffer 1: flush output buffer 2: flush input & output buffer

Description

flush the driver's input/output buffer; the data will no longer exist

Return Value

SIO_OK	OK
SIO_BADPORT	port number is invalid
SIO_BADPARM	bad parameter

sio_getbaud	get serial port's baud rate setting	**Port Status**

Syntax

```
#include  <sdksio.h>
long  sio_getbaud ( int port );
```

Arguments

port	async serial port number

Description

get the serial port's baud rate setting; the return value is the actual baud rate; a return value of **9600** means 9600 bps, and a return value of **200** means 200 bps

Return Value

> 0	baud rate
SIO_BADPORT	port number is invalid

sio_getch	read one character from driver's input buffer	**Data Input**

Syntax

```
#include  <sdksio.h>
int  sio_getch ( int port );
```

Arguments

port	async serial port number

Description

read one character from the driver's input buffer

Return Value

0 to 255	ASCII code of the character received
SIO_BADPORT	port number is invalid
SIO_NODATA	no data to read
SIO_BADPARM	bad parameter
SIO_NOT_OPEN	port was not open in advance

`sio_getflow`	get serial port's hardware and software flow control settings	**Port Status**

Syntax

```
#include  <sdksio.h>
int  sio_getflow ( int port );
```

Arguments

`port`	async serial port number

Description

get the serial port's hardware and software flow control settings; refer to **`sio_flowctrl()`** for detail

Return Value

`>=0`	bit 0: l CTS flow control bit 1: RTS flow control bit 2: Tx XON/XOFF flow control bit 3: Rx XON/XOFF flow control
`SIO_BADPORT`	port number is invalid

`sio_getmode`	get serial port’s mode settings	**Port Status**

Syntax

```
#include  <sdksio.h>
int  sio_getmode ( int port );
```

Arguments

`port`	async serial port number

Description

get the serial port’s mode settings; refer to the description of **`sio_ioctl()`** to see the mode settings

Return Value

`>=0`	the mode settings, see **`sio_ioctl()`** for detail
`SIO_BADPORT`	port number is invalid

`sio_GetReadTimeouts`	get timeout values for **`sio_read()`** and **`sio_getch()`**	**Data Input**

Syntax

```
#include  <sdksio.h>
int  sio_GetReadTimeouts ( int port, DWORD *TotalTimeouts, DWORD
*IntervalTimeouts );
```

Arguments

`port`	async serial port number
`TotalTimeouts`	a pointer to buffer to retrieve total timeout value
`IntervalTimeouts`	a pointer to buffer to retrieve interval timeout value

Description

get timeout values for **`sio_read()`** and **`sio_getch()`**

Return Value

`SIO_OK`	OK
`SIO_BADPORT`	port number is invalid

sio_GetWriteTimeouts	get timeout value for **sio_write()** and **sio_putch()**	**Data Output**

Syntax

```
#include  <sdksio.h>
int  sio_GetWriteTimeouts ( int port, DWORD *TotalTimeouts );
```

Arguments

port	async serial port number
TotalTimeouts	a pointer to buffer to retrieve the total timeout value

Description

get timeout values for **sio_write()** and **sio_putch()**

Return Value

SIO_OK	OK
SIO_BADPORT	port number is invalid

sio_ioctl	modify settings of serial port's I/O control register	**Port Control**

Syntax

```
#include  <sdksio.h>
int  sio_ioctl ( int port, int baud, int mode );
```

Arguments

port	async serial port number		
baud	(bits/sec)		
	0: 50	6: 600	12: 9600
	1: 75	7: 1200	13: 19200
	2: 110	8: 1800	14: 38400
	3: 134.5	9: 2400	15: 57600
	4: 150	10: 4800	16: 115200
	5: 300	11: 7200	17: 230400

mode — **bit_cnt** OR **stop_bit** OR **parity**

bit_cnt (bits 0-1)
- 0x00: data bit 5
- 0x01: data bit 6
- 0x02: data bit 7
- 0x03: data bit 8

stop_bit (bit 2)
- 0x00: stop bit 1
- 0x04: stop bit 1.5 or 2

parity (bits 3- 5)
- 0x00: no parity
- 0x08: odd parity
- 0x18: even parity
- 0x28: mark parity
- 0x38: space parity

Description

modify the settings of the serial port's I/O control register, such as baud rate, parity, data bits, and stop bit

Return Value

SIO_OK	OK
SIO_BADPORT	port number is invalid
SIO_BADPARM	bad parameter
SIO_NOT_OPEN	port was not open in advance

`sio_iqueue`	get length of data accumulated in system and driver input buffers	**Port Status**

Syntax

```
#include  <sdksio.h>
long  sio_iqueue ( int port );
```

Arguments

`port`	async serial port number

Description

get the length of data accumulated in the system's input buffer and driver's input buffer; note that even when **`sio_iqueue()`** returns a zero value, there may be a few characters remaining in the RS-232 UART chip that are not yet known

Return Value

`>= 0`	data currently in input buffer (bytes)
`SIO_BADPORT`	port number is invalid

`sio_lctrl`	set DTR and RTS states	**Port Control**

Syntax

```
#include  <sdksio.h>
int  sio_lctrl ( int port, int mode );
```

Arguments

`port`	async serial port number
`mode`	**`C_DTR`** (bit 0)
	`C_RTS` (bit 1)

Description

set both the DTR and RTS states

Return Value

`SIO_OK`	OK
`SIO_BADPORT`	port number is invalid
`SIO_BADPARM`	bad parameter
`SIO_NOT_OPEN`	port was not open in advance
`SIO_RTS_BY_HW`	cannot control the port because it is set as auto hardware flow control by **`sio_flowctrl()`**

sio_linput	read a block of data ending with termination character	**Data Input**

Syntax

```
#include  <sdksio.h>
int  sio_linput ( int port, char *buf, int len, int term );
```

Arguments

port	async serial port number
buf	receive buffer pointer
len	buffer length (bytes)
term	terminator code

Description

read a block of data from the driver's input buffer until the terminator character is encountered or **len** bytes of data are read

Return Value

> 0	length of data received (bytes)
= 0	no data received
SIO_BADPORT	port number is invalid
SIO_BADPARM	bad parameter
SIO_NOT_OPEN	port was not open in advance

sio_lstatus	get status of the serial line	**Port Status**

Syntax

```
#include  <sdksio.h>
int  sio_lstatus ( int port );
```

Arguments

port	async serial port number

Description

get the status of the line

Return Value

>= 0	current line status bit 0: **S_CTS** bit 1: **S_DSR** bit 2: **S_RI** bit 3: **S_CD**
SIO_BADPORT	port number is invalid

`sio_modem_irq`	set event service routine to be called when line status has changed	**Event Control**

Syntax

```
#include  <sdksio.h>
int  sio_modem_irq ( int port, void (*func) (int port) );
```

Arguments

`port`	async serial port number
`func`	event service routine entry; if the **`func`** is NULL, it will disable this routine

Description

set event service routine to be called when line status has changed; when line status (CTS, DSR, CD, RI) changes, the system will call the event service routine

Return Value

`SIO_OK`	OK
`SIO_BADPORT`	port number is invalid
`SIO_NOT_OPEN`	port was not open in advance

`sio_ofree`	get amount of free space in driver's output buffer	**Port Status**

Syntax

```
#include  <sdksio.h>
long  sio_ofree ( int port );
```

Arguments

`port`	async serial port number

Description

get the amount of free space in the driver's output buffer

Return Value

`>= 0`	amount of free space in output buffer (bytes)
`SIO_BADPORT`	port number is invalid

`sio_open`	enable serial port to transmit and receive data	**Port Control**

Syntax

```
#include  <sdksio.h>
int  sio_open ( int port );
```

Arguments

`port`	async serial port number

Description

enable a serial port to transmit and receive data; after calling **`sio_open()`**, the initial status of this serial port is the same as the last setting or configuration setting

Return Value

`>= 0`	indicates successful open action, and return value is a descriptor referencing the port; programmer can use this descriptor in the **`select()`** function (from the socket API group) to carry out a data read/write operation
`SIO_BADPORT`	port number is invalid

`sio_oqueue`	get the amount of data remaining in system and driver output buffers	**Port Status**

Syntax

```
#include  <sdksio.h>
long  sio_oqueue ( int port );
```

Arguments

`port`	async serial port number

Description

get the amount of data not yet sent out in both the system's output buffer and the driver's output buffer

Return Value

`>= 0`	amount of data (bytes), still remaining in driver's output buffer
`SIO_BADPORT`	port number is invalid

`sio_putch`	write a character into driver's output buffer	**Data Output**

Syntax

```
#include  <sdksio.h>
int  sio_putch ( int port, int term );
```

Arguments

`port`	async serial port number
`term`	the character to be written

Description

write a character into driver's output buffer

Return Value

`SIO_OK`	OK
`SIO_BADPORT`	port number is invalid
`SIO_BADPARM`	bad parameter
`SIO_ABORT_WRITE`	user abort blocked write
`SIO_WRITETIMEOUT`	write timeout has occurred
`SIO_NOT_OPEN`	port was not open in advance

sio_read	read data from the driver's input buffer	**Data Input**

Syntax

```
#include  <sdksio.h>
int  sio_read ( int port, char *buf, int len );
```

Arguments

port	async serial port number
buf	receive buffer pointer
len	buffer length (bytes)

Description

sio_read() reads data from the driver's input buffer. If the user's buffer is large enough to hold the data in the driver's input buffer, then the entire contents of the driver's input buffer will be transferred to the user's buffer. Otherwise, only **len** bytes will be transferred to the user's buffer.

sio_SetReadTimeout() can be used to set timeouts for **sio_read()**.
sio_AbortRead() can be used to abort any blocked **sio_read()**.

Return Value

> 0	length of data received (bytes)
= 0	no data received
SIO_BADPORT	port number is invalid
SIO_BADPARM	bad parameter
SIO_NOT_OPEN	port was not open in advance

sio_RTS	set RTS state of port	**Port Control**

Syntax

```
#include  <sdksio.h>
int  sio_RTS ( int port, int mode );
```

Arguments

port	async serial port number
mode	0: turn RTS off 1: turn RTS on

Description

set RTS state of port

Return Value

SIO_OK	OK
SIO_BADPORT	port number is invalid
SIO_BADPARM	bad parameter
SIO_RTS_BY_HW	cannot control the port because it is set as auto H/W flow control by **sio_flowctrl()**
SIO_NOT_OPEN	port was not open in advance

sio_SetReadTimeouts	set timeout values for **sio_read()** and **sio_getch()**	**Data Input**

Syntax

```
#include  <sdksio.h>
int  sio_SetReadTimeouts ( int port, DWORD TotalTimeouts, DWORD
IntervalTimeouts );
```

Arguments

port	async serial port number
TotalTimeouts	total timeout values (milliseconds)
IntervalTimeouts	interval timeout values (milliseconds)

Description

set timeout values for **sio_read()** and **sio_getch()**; the default **TotalTimeouts** value is 0xFFFFFFFF and the default **IntervalTimeouts** value is 0, which enables **sio_read()** to return immediately

Return Value

SIO_OK	OK
SIO_BADPORT	port number is invalid

sio_SetWriteTimeouts	set timeout value for **sio_write()** and **sio_putch()**	**Data Output**

Syntax

```
#include  <sdksio.h>
int  sio_SetWriteTimeouts ( int port, DWORD TotalTimeouts );
```

Arguments

port	async serial port number
TotalTimeouts	total timeout value (milliseconds)

Description

set timeout value for **sio_write()** and **sio_putch(**); the default value of write timeout is 0xFFFFFFFF, which enables **sio_write()** and **sio_putch()** to return immediately without blocking at all

the value 0 enables **sio_write(**) to always block until finished writing data

Return Value

SIO_OK	OK
SIO_BADPORT	port number is invalid
SIO_BADPARM	bad parameter

sio_term_irq	set an event service routine to be called when the terminator character is received	**Event Control**

Syntax

```
#include  <sdksio.h>
int  sio_term_irq ( int port, void (*func) (int port),
char code );
```

Arguments

port	async serial port number
func	event service routine entry; if the **func** is NULL, it will disable this routine
code	terminator character code

Description

set an event service routine to be called when the terminator character is received; when the terminator character is received, the system will call the event service routine

Return Value

SIO_OK	OK
SIO_BADPORT	port number is invalid
SIO_NOT_OPEN	port was not open in advance

sio_Tx_empty_irq	set an event service routine to be called when output buffer is cleared	**Event Control**

Syntax

```
#include  <sdksio.h>
int  sio_Tx_empty_irq ( int port, void (*func) (int port) );
```

Arguments

port	async serial port number
func	event service routine entry; if the **func** is NULL, it will disable this routine

Description

set an event service routine to be called when last character in output buffer is sent; when the Tx empty signal is encountered, the system will call the event service routine

Return Value

SIO_OK	OK
SIO_BADPORT	port number is invalid
SIO_NOT_OPEN	port was not open in advance

sio_Tx_hold	check why data could not be transmitted	**Port Status**

Syntax

```
#include  <sdksio.h>
int  sio_Tx_hold ( int port );
```

Arguments

port	async serial port number

Description

check the reason why data could not be transmitted

Return Value

>=0	bit 0 on: data could not transmitted because CTS is low bit 1 on: data could not transmitted because XOFF char received
SIO_BADPORT	port number is invalid

sio_write	write a block of data to driver's output buffer	**Data Output**

Syntax

```
#include  <sdksio.h>
int  sio_write ( int port, char *buf, int len );
```

Arguments

port	async serial port number
buf	transmit string pointer
len	transmit string length (bytes)

Description

sio_write writes a block of data to the driver's output buffer. The actual length of data written depends on the amount of free space in the driver's output buffer. **sio_write()** is always non-block by default.

Use **sio_SetWriteTimeout()** to set the timeout for **sio_write()**. **SIO_WRITETIMEOUT** will be returned from **sio_write()** when the write function times out.

sio_AbortWrite() can be used to abort any blocked **sio_write()** with return value **SIO_ABORT_WRITE**.

Return Value

>= 0	length of data transmitted (bytes)
SIO_BADPORT	port number is invalid
SIO_BADPARM	bad parameter
SIO_ABORT_WRITE	user abort blocked write
SIO_WRITETIMEOUT	write timeout has occurred
SIO_NOT_OPEN	port was not open in advance

BSD Socket Library Reference

accept	accept a connection on a socket	**Socket Control**

Syntax

```
#include  <sdksock.h>
int  accept ( int s, SOCKADDR *addr, int *addrlen );
```

Arguments

s	a descriptor identifying a socket which is listening for connections after a **listen()**
addr	an optional pointer to a buffer that receives the address of the connecting entity, as known to the communications layer; exact format of the **addr** argument is determined by the address family established when the socket was created
addrlen	an optional pointer to an integer that contains the length of the address **addr**

Description

This routine extracts the first connection on the queue of pending connections on **s**, creates a new socket with the same properties as **s**, and returns a handle to the new socket. If no pending connections are present on the queue and the socket is not marked as non-blocking, **accept()** blocks the caller until a connection is present. If the socket is marked non-blocking and no pending connections are present on the queue, **accept()** returns an error as described below. The accepted socket may not be used to accept more connections. The original socket remains open.

The argument **addr** is a result parameter that is filled in with the address of the connecting entity, as known to the communications layer. The exact format of the **addr** parameter is determined by the address family in which the communication is occurring. The **addrlen** is a value-result parameter; it should initially contain the amount of space pointed to by **addr**. On return, it will contain the actual length (bytes), of the address returned. This call is used with connection-based socket types such as **SOCK_STREAM**. If **addr** or **addrlen** are equal to NULL, then no information about the remote address of the accepted socket is returned.

Return Value

If there are no errors, **accept()** returns the descriptor for the accepted packet. Otherwise, a value of -1 is returned, and the global variable **errno** will contain one of the following values.

Error Codes

EBADF	the first argument does not specify a valid descriptor
EOPNOTSUPP	the socket is not of type **SOCK_STREAM**
EFAULT	the pointer in one of the arguments is invalid
EWOULDBLOCK	the socket is marked non-blocking and no connections are waiting to be accepted
EFILE	the initial system file table is full

bind	associate a local address with a socket	**Socket Control**

Syntax

```
#include  <sdksock.h>
int  bind ( int s, SOCKADDR *name, int namelen );
```

Arguments

s	a descriptor identifying an unbound socket
name	the address to assign to the socket
namelen	length of the value in **name**

Description

This routine is used on an unconnected datagram or stream socket, before subsequent **connect()** or **listen()** routines. When a socket is created with **socket()**, it exists in a name space (address family), but it has no name assigned. **bind()** establishes the local association (host address/port number) of the socket by assigning a local name to an unnamed socket.

In the Internet address family, a name consists of several components. For **SOCK_DGRAM** and **SOCK_STREAM**, the name consists of three parts: a host address, the protocol number, and a port number which identifies the application. If an application does not care what address is assigned to it, it may specify an Internet address equal to **INADDR_ANY**, a port equal to 0, or both. If the Internet address is equal to **INADDR_ANY**, any appropriate network interface will be used; this simplifies application programming in the presence of multi-homed hosts. If the port is specified as 0, the implementation will assign a unique port to the application with a value between 1024 and 30000. The application may use **getsockname()** after **bind()** to retrieve the address that has been assigned to it, but note that **getsockname()** will not necessarily fill in the Internet address until the socket is connected, since several Internet addresses may be valid if the host is multi-homed.

Return Value

If there are no errors, **bind()** returns 0. Otherwise, it returns -1, and the global variable **errno** will contain one of the following values.

Error Codes

EFAULT	the **namelen** argument is too small (less than the size of a **SOCKADDR**) or the **name** argument pointer is invalid
EINVAL	the socket is already bound to an address
EBADF	the descriptor is not a socket

`closesocket`	close a socket	**Socket Control**

Syntax

```
#include  <sdksock.h>
int  closesocket ( int s );
```

Arguments

`s`	a descriptor identifying a socket

Description

This function closes a socket. More precisely, it releases the socket descriptor s, so that further references to **`s`** will fail with the error **`EBADF`**. If this is the last reference to the underlying socket, the associated naming information and queued data are discarded.

The semantics of **`closesocket()`** are affected by the socket options **`SO_LINGER`** and **`SO_DONTLINGER`** as follows:

Option	**Interval**	**Type of close**	**Wait for close?**
`SO_DONTLINGER`	don’t care	graceful	no
`SO_LINGER`	zero	hard	no
`SO_LINGER`	non-zero	graceful	yes

If **`SO_LINGER`** is set (i.e., the **`l_onoff`** field of the linger structure is non-zero) with a zero timeout interval (**`l_linger`** is zero), **`closesocket()`** is not blocked even if queued data has not yet been sent or acknowledged. This is called a “hard” or “abortive” close, because the socket’s virtual circuit is reset immediately, and any unsent data is lost.

If **`SO_LINGER`** is set with a non-zero timeout interval, the **`closesocket()`** call blocks until the remaining data has been sent or until the timeout expires. This is called a graceful disconnect.

If **`SO_DONTLINGER`** is set on a stream socket (i.e. the **`l_onoff`** field of the **`linger`** structure is zero), the **`closesocket()`** call will return immediately. However, any data queued for transmission will be sent if possible before the underlying socket is closed. This is also called a graceful disconnect. Note that in this case the implementation may not release the socket and other resources for an arbitrary period, which may affect applications which expect to use all available sockets.

Return Value

If there are no errors, **`closesocket()`** returns 0. Otherwise, it returns -1, and the global variable **`errno`** will contain one of the following values.

Error Codes

`EBADF`	the descriptor is not a socket

connect	establish a connection to a peer	**Socket Control**

Syntax

```
#include  <sdksock.h>
int  connect ( int s, SOCKADDR *name, int namelen );
```

Arguments

s	a descriptor identifying an unconnected socket
name	the name of the peer to which the socket is to be connected
namelen	length of the value in **name**

Description

This function is used to create a connection to the specified foreign association. The parameter **s** specifies an unconnected datagram or stream socket. If the socket is unbound, unique values are assigned to the local association by the system, and the socket is marked as bound. Note that if the address field of the name structure is all zeroes, **connect()** will return the error **EADDRNOTAVAIL**.

For stream sockets (type **SOCK_STREAM**), an active connection is initiated to the foreign host using **name** (an address in the name space of the socket). When the socket call completes successfully, the socket is ready to send and receive data.

For a datagram socket (type **SOCK_DGRAM**), a default destination is set, which will be used on subsequent **send()** and **recv()** calls.

Return Value

On a blocking socket, the return value indicates success or failure of the connection attempt.

On a non-blocking socket, if the return value is -1, an application should check **errno**. If this indicates an error code of **EINPROGRESS**, then your application can use **select()** to determine the completion of the connection request by checking if the socket is writeable.

Error Codes

EINPROGRESS	(TCP only) socket is nonblocking and a connection attempt would block
EADDRNOTAVAIL	specified address is not available
EADDRINUSE	specified address already in use
ECONNREFUSED	(TCP only) attempt to connect was forcefully rejected by the remote machine
EISCONN	socket is already connected
EBADF	descriptor is not a socket
ETIMEDOUT	(TCP only) attempt to connect timed out without establishing a connection; current timeout value is 30 seconds

gethostbyname	get host information corresponding to a hostname	**Inquiry**

Syntax

```
#include  <sdksock.h>
struct hostent  *gethostbyname ( char *name );
```

Arguments

name	a pointer to the name of the host

Description

gethostbyname() returns a pointer to the following structure which contains the name(s) and address that correspond to the given address.

```
struct hostent {
    char *    h_name;
    char **   h_aliases;
    short     h_addrtype;
    short     h_length;
    char **   h_addr_list;
};
```

The members of this structure are:

Element	Usage
h_name	server name of local system
h_aliases	a NULL-terminated array of alternate names, currently unused
h_addrtype	the type of address being returned; this is always **AF_INET**
h_length	the length (bytes); this is always 4
h_addr_list	a NULL-terminated list of addresses for the host addresses, returned in network byte order

The pointer returned points to a structure that is allocated by the NE-4100-P. Applications must not modify this structure or free any of its components.

Return Value

If there are no errors, **gethostbyname()** returns a pointer to the **hostent** structure described above. Otherwise, it returns a NULL pointer.

gethostname	return the standard host name for the local machine	**Inquiry**

Syntax

```
#include  <sdksock.h>
int  gethostname ( char *name, int namelen );
```

Arguments

name	a pointer to a buffer that will receive the host name
namelen	length of the buffer

Description

This routine returns the name of the local host into the buffer specified by the **name** parameter. The host name is returned as a null-terminated string. The form of the host name is dependent on the socket's implementation—it is a simple host name. However, it is guaranteed that the name returned will be successfully parsed by **gethostbyname()**.

Return Value

If there are no errors, **gethostname()** returns 0. Otherwise, it returns -1, and the global variable **errno** will contain one of the following values.

Error Codes

EFAUL	**namelen** parameter is too small or **name** pointer is invalid

getpeername	get the address of the peer to which a socket is connected	**Inquiry**

Syntax

```
#include  <sdksock.h>
int  getpeername ( int s, SOCKADDR *name, int *namelen );
```

Arguments

s	a descriptor identifying a connected socket
name	the structure which is to receive the name of the peer
namelen	a pointer to the length of the **name** structure

Description

getpeername() retrieves the name of the peer connected to the socket **s** and stores it in the **SOCKADDR** identified by **name**. It is used on a connected datagram or stream socket.

On return, the **namelen** argument contains the actual size of the name returned (bytes).

Return Value

If there are no errors, **getpeername()** returns 0. Otherwise, it returns -1, and the global variable **errno** will contain one of the following values.

Error Codes

EFAULT	**namelen** parameter is too small or **name** pointer is invalid
ENOTCONN	socket is not connected
EBADF	descriptor is not a socket

getsockname	get the local name for a socket	**Inquiry**

Syntax

```
#include  <sdksock.h>
int  getsockname ( int s, SOCKADDR *name, int *namelen );
```

Arguments

s	a descriptor identifying a bound socket
name	receives the address (name) of the socket
namelen	the length of the **name** buffer

Description

getsockname() retrieves the current name for the specified socket descriptor in **name**. It is used on a bound and/or connected socket specified by the **s** parameter. The local association is returned. This call is especially useful when a **connect**() call has been made without first doing a **bind()**; this call provides the only means by which you can determine the local association which has been set by the system.

On return, the **namelen** argument contains the actual size of the name returned (bytes).

If a socket was bound to **INADDR_ANY**, indicating that any of the host's IP addresses should be used for the socket, **getsockname()** will not necessarily return information about the host IP address, unless the socket has been connected with **connect()** or **accept()**.

Return Value

If there are no errors, **getsockname()** returns 0. Otherwise, it returns -1, and the global variable **errno** will contain one of the following values.

Error Codes

EFAULT	address of **name** or **namelen** argument is not large enough
EBADF	descriptor is not a socket

getsockopt	retrieve a socket option	**Socket Control**

Syntax

```
#include  <sdksock.h>
int  getsockopt ( int s, int level, int optname, char *optval, int
*optlen );
```

Arguments

s	a descriptor identifying a socket
level	the level at which the option is defined; the only supported levels are **SOL_SOCKET**
optname	the socket option for which the value is to be retrieved
optval	a pointer to the buffer in which the value for the requested option is to be returned
optlen	a pointer to the size of the **optval** buffer

Description

getsockopt() retrieves the current value for a socket option associated with a socket of any type, in any state, and stores the result in **optval**. Options may exist at multiple protocol levels, but they are always present at the uppermost “socket’’ level.

The value associated with the selected option is returned in the buffer **optval**. The integer pointed to by **optlen** should originally contain the size of this buffer; on return, it will be set to the size of the value returned. For **SO_LINGER**, this will be the size of a struct linger; for all other options it will be the size of an integer.

If the option was never set with **setsockopt()**, then **getsockopt()** returns the default value for the option.

The following socket options are supported for **getsockopt()**. The Type identifies the type of data addressed by **optval**:

Value	Type	Meaning
SO_DONTLINGER	BOOL	if true, the **SO_LINGER** option is disabled
SO_KEEPALIVE	BOOL	keepalives are being sent
SO_LINGER	LINGER	returns the current linger options

Calling **getsockopt()** with an unsupported option will result in an error code of **ENOPROTOOPT**.

Return Value

If there are no errors, **getsockopt()** returns 0. Otherwise, it returns -1, and the global variable **errno** will contain one of the following values.

Error Codes

EFAULT	**optlen** argument was invalid
ENOPROTOOPT	option is unknown or unsupported
EBADF	descriptor is not a socket

htonl	convert an unsigned long from host to network byte order	**Misc.**

Syntax

```
#include  <sdksock.h>
u_long  htonl ( u_long hostlong );
```

Arguments

hostlong	a 32-bit number in host byte order

Description

This routine takes a 32-bit number in host byte order and returns a 32-bit number in network byte order.

Return Value

htonl() returns the value in network byte order.

htons	convert an unsigned short from host to network byte order	**Misc.**

Syntax

```
#include  <sdksock.h>
u_short  htons ( u_short hostshort );
```

Arguments

hostshort	a 16-bit number in host byte order

Description

This routine takes a 16-bit number in host byte order and returns a 16-bit number in network byte order.

Return Value

htons() returns the value in network byte order.

inet_addr	convert a string containing a dotted address into an long integer	**Misc.**

Syntax

```
#include  <sdksock.h>
unsigned long inet_addr ( char *cp );
```

Arguments

cp	character string of an Internet address in standard "." notation

Description

This function interprets the character string specified by the **cp** parameter. This string represents a numeric Internet address expressed in the Internet standard "." notation. The value returned is a number suitable for use as an Internet address. All Internet addresses are returned in network order (bytes ordered from left to right).

Internet addresses specified using the "." notation take the form **a.b.c.d**, such as **192.168.127.254**.

When four parts are specified, each is interpreted as a byte of data and assigned, from left to right, to the four bytes of an Internet address. Note that when an Internet address is viewed as a 32-bit integer quantity on the Intel architecture, the bytes referred to above appear as **d.c.b.a**. That is, the bytes on an Intel processor are ordered from right to left.

Return Value

If there are no errors, **inet_addr()** returns an unsigned long containing a suitable binary representation of the Internet address given. If the passed-in string does not contain a legitimate Internet address, for example if a portion of an "a.b.c.d" address exceeds 255, **inet_addr()** returns the value **INADDR_ANY**.

inet_ntoa	convert a network address into a string in dotted format	**Misc.**

Syntax

```
#include  <sdksock.h>
char  *inet_ntoa ( unsigned long in );
```

Arguments

in	an Internet host address

Description

This function takes an Internet address specified by the **in** parameter. It returns an ASCII string representing the address in "." notation as **a.b.c.d**. Note that the string returned by **inet_ntoa()** resides in memory which is allocated by the sockets implementation. The application should not make any assumptions about the way in which the memory is allocated. The data is guaranteed to be valid until the next socket API call, but no longer.

Return Value

If there are no errors, **inet_ntoa()** returns a character pointer to a static buffer containing the text address in standard "." notation. Otherwise, it returns NULL. The data should be copied before another sockets call is made.

`ioctlsocket`	control the mode of a socket	**Socket Control**

Syntax

```
#include  <sdksock.h>
int  ioctlsocket ( int s, long cmd, u_long *argp );
```

Arguments

`s`	a descriptor identifying a socket
`cmd`	the command to perform on the socket **`s`**
`argp`	a pointer to a parameter for **`cmd`**

Description

This routine may be used on any socket in any state. It is used to get or retrieve operating parameters associated with the socket, independent of the protocol and communication subsystem. The following commands are supported:

Command	**Semantics**
`FIONBIO`	Use this command to enable or disable non-blocking mode on socket **`s`**. The parameter **`argp`** points to an unsigned long, which is non-zero if non-blocking mode is to be enabled and zero if it is to be disabled. When a socket is created, it operates in blocking mode (i.e., non-blocking mode is disabled).

Return Value

Upon successful completion, the **`ioctlsocket()`** returns 0. Otherwise, it returns -1, and the global variable **`errno`** will contain one of the following values.

Error Codes

`EINVAL`	**`cmd`** is not a valid command, **`argp`** is not an acceptable parameter for **`cmd`**, or the command is not applicable to the type of socket supplied
`EBADF`	descriptor **`s`** is not a socket

`listen`	establish a socket to listen for incoming connection	**Socket Control**

Syntax

```
#include  <sdksock.h>
int  listen ( int s, int backlog );
```

Arguments

`s`	descriptor identifying a bound, unconnected socket
`backlog`	maximum number of connections that can be established from the socket; this is different from the standard BSD socket

Description

To accept connections, a socket is first created with **`socket()`**. A backlog for incoming connections is then specified with **`listen()`**, and then connections are accepted with **`accept()`**. **`listen()`** applies only to sockets that support connections (i.e., those of type **`SOCK_STREAM`**). The socket **`s`** is put into "passive" mode where incoming connections are acknowledged and queued pending acceptance by the process.

Return Value

If there are no errors, **`listen()`** returns 0. Otherwise, it returns -1, and the global variable **`errno`** will contain one of the following values.

Error Codes

`EBADF`	descriptor is not a socket
`EOPNOTSUPP`	referenced socket is not of a type that supports the **`listen()`** operation

`ntohl`	convert an unsigned long from network to host byte order	**Misc.**

Syntax

```
#include  <sdksock.h>
u_long  ntohl ( u_long netlong );
```

Arguments

`netlong`	a 32-bit number in network byte order

Description

This routine takes a 32-bit number in network byte order and returns a 32-bit number in host byte order.

Return Value

`ntohl()` returns the value in host byte order.

ntohs	convert an unsigned short from network to host byte order.	**Misc.**

Syntax

```
#include  <sdksock.h>
u_short  ntohs ( u_short netshort );
```

Arguments

netshort	a 16-bit number in network byte order

Description

This routine takes a 16-bit number in network byte order and returns a 16-bit number in host byte order.

Return Value

ntohs() returns the value in host byte order.

recv	receive data from a socket	**Data Input/Output**

Syntax

```
#include  <sdksock.h>
int  recv ( int s, char *buf, int len, int flags );
```

Arguments

s	a descriptor identifying a connected socket
buf	a buffer for the incoming data
len	the size of buffer pointed by **buf**
flags	specifies the way in which the call is made

Description

This function is used on datagram or connected stream sockets specified by the **s** parameter and is used to read incoming data.

For sockets of type **SOCK_STREAM**, all information currently available up to the size of the buffer supplied is returned.

For datagram sockets, data is extracted from the first enqueued datagram, up to the size of the buffer supplied. If the datagram is larger than the buffer supplied, the buffer is filled with the first part of the datagram, and the excess data is lost.

If no incoming data is available at the socket, the **recv()** call waits for data to arrive unless the socket is non-blocking. In this case a value of -1 is returned with the error code set to **EWOULDBLOCK**. The **select()** calls may be used to determine when more data arrives.

If the socket is of type **SOCK_STREAM** and the remote side has shut down the connection gracefully or the connection has been reset, a **recv()** will complete immediately with 0 bytes received.

flags may be used to influence the behavior of the function invocation beyond the options specified for the associated socket. That is, the semantics of this function are determined by the socket options and the **flags** parameter. The latter is constructed by "or-ing" any of the following values:

Value	Meaning
MSG_OOB	read out-of-band data (**SOCK_STREAM** only)

Return Value

If there are no errors, **recv()** returns the number of bytes received. If the connection has been closed, it returns 0. Otherwise, it returns -1, and the global variable **errno** will contain one of the following values.

Error Codes

EBADF	descriptor is not a socket
EFAULT	**buf** argument pointer is invalid
EOPNOTSUPP	**MSG_OOB** was specified, but the socket is not of type **SOCK_STREAM**
ESHUTDOWN	socket has been shutdown; it is not possible to **recv()** on a socket after **shutdown()** has been invoked with **how** set to 0 or 2
EWOULDBLOCK	socket is marked as non-blocking and the receive operation would block
EIO	**MSG_OOB** was specified, but has not received out-of-band data
ELENZERO	**length** argument is zero

`recvfrom`	receive a datagram and store the source address	**Data Input/Output**

Syntax

```
#include  <sdksock.h>
int  recvfrom ( int s, char *buf, int len, int flags, SOCKADDR *from,
int *fromlen );
```

Arguments

`s`	a descriptor identifying a bound socket
`buf`	a buffer for the incoming data
`len`	the length of **`buf`**
`flags`	specifies the way in which the call is made
`from`	an optional pointer to a buffer which will hold the source address upon return
`fromlen`	an optional pointer to the size of the **`from`** buffer

Description

This function is used to read incoming data on a (possibly connected) socket and capture the address from which the data was sent.

For sockets of type **`SOCK_STREAM`**, all information currently available up to the size of the buffer supplied is returned. The **`from`** and **`fromlen`** parameters are ignored for **`SOCK_STREAM`** sockets.

For datagram sockets, data is extracted from the first enqueued datagram, up to the size of the buffer supplied. If the datagram is larger than the buffer supplied, the buffer is filled with the first part of the message, and the excess data is lost.

If **`from`** is non-zero, and the socket is of type **`SOCK_DGRAM`**, the network address of the peer which sent the data is copied to the corresponding **`SOCKADDR`**. The value pointed to by **`fromlen`** is initialized to the size of this structure, and is modified on return to indicate the actual size of the address stored there.

If no incoming data is available at the socket, the **`recvfrom()`** call waits for data to arrive unless the socket is non-blocking. In this case a value of -1 is returned with the error code set to **`EWOULDBLOCK`**. The **`select()`** calls may be used to determine when more data arrives.

If the socket is of type **`SOCK_STREAM`** and the remote side has shut down the connection gracefully or the connection has been reset, a **`recvfrom()`** will complete immediately with 0 bytes received.

`flags` may be used to influence the behavior of the function invocation beyond the options specified for the associated socket. That is, the semantics of this function are determined by the socket options and the **`flags`** parameter. The latter is constructed by "or-ing" any of the following values:

Value	**Meaning**
`MSG_OOB`	read out-of-band data (**`SOCK_STREAM`** only)

Return Value

If there are no errors, **`recvfrom()`** returns the number of bytes received. If the connection has been closed, it returns 0. Otherwise, it returns -1, and the global variable **`errno`** will contain one of the following values.

`recvfrom`	receive a datagram and store the source address	**Data Input/Output**
Error Codes		
`EBADF`	descriptor is not a socket	
`EFAULT`	**`buf`** argument pointer is invalid	
`EOPNOTSUPP`	**`MSG_OOB`** was specified, but socket is not of type **`SOCK_STREAM`**	
`ESHUTDOWN`	socket has been shut down; it is not possible to **`recvfrom()`** on a socket after **`shutdown()`** has been invoked with **`how`** set to 0 or 2	
`EWOULDBLOCK`	socket is marked as non-blocking and the receive operation would block	
`EIO`	**`MSG_OOB`** was specified, but has not received out-of-band data	
`ELENZERO`	**`fromlen`** argument is zero	

`select`	determine the status of one or more sockets, waiting if necessary	**Data Input/Output**

Syntax

```
#include  <sdksock.h>
int  select ( int nfds, fd_set *readfds, fd_set *writefds, fd_set
*exceptfds, struct timeval *timeout );
```

Arguments

`nfds`	indicates the range of sockets to be checked
`readfds`	an optional pointer to a set of sockets to be checked for readability
`writefds`	an optional pointer to a set of sockets to be checked for writeability
`exceptfds`	an optional pointer to a set of sockets to be checked for errors
`timeout`	the maximum time for **`select()`** to wait, or NULL for blocking operation

Description

This function is used to determine the status of one or more sockets. For each socket, the caller may request information on read, write or error status. The set of sockets for which a given status is requested is indicated by an **`fd_set`** structure. Upon return, the structure is updated to reflect the subset of these sockets which meet the specified condition, and **`select()`** returns the number of sockets meeting the conditions. A set of macros is provided for manipulating an **`fd_set`**. These macros are compatible with those used in the Berkeley software, but the underlying representation is completely different.

In each set of sockets, the descriptors from 0 through **`nfds`**-1 will be examined. This value should not exceed the number of sockets the system allows. For the value of **`nfds`**, we recommend that you use the maximum number within the socket sets and add 1. For example, if the maximum number among the socket sets is 7, then 8 should be used as for **`nfds`**.

Three independent sets of descriptors are watched. Those listed in **`readfds`** will be watched to see if characters become available for reading, those in **`writefds`** will be watched to see if it is OK to immediately write on them, and those in **`exceptfds`** will be watched for exceptions. On exit, the sets are modified in place to indicate which descriptors actually changed status.

Any of **`readfds`**, **`writefds`**, or **`exceptfds`** may be given as NULL if no descriptors are of interest.

Four macros are defined in the header file **`sdksock.h`** for manipulating the descriptor sets. The variable **`FD_SETSIZE`** determines the maximum number of descriptors in a set (the default value of **`FD_SETSIZE`** is 96). Internally, an **`fd_set`** is represented as an array of int's. The macros are as follows:

`FD_CLR(s, *set)`	removes the descriptor **`s`** from set.
`FD_ISSET(s, *set)`	nonzero if **`s`** is a member of the set, or zero otherwise
`FD_SET(s, *set)`	adds descriptor **`s`** to set
`FD_ZERO(*set)`	initializes the set to the NULL set

The parameter **`timeout`** controls how long the **`select()`** may take to complete. If **`timeout`** is a null pointer, **`select()`** will block indefinitely until at least one descriptor meets the specified criteria. Otherwise, **`timeout`** points to a struct timeval which specifies the maximum time that **`select()`** should wait before returning. If the timeval is initialized to {0, 0}, **`select()`** will return immediately; this is used to "poll" the state of the selected sockets.

`select`	determine the status of one or more sockets, waiting if necessary	**Data Input/Output**

Return Value

`select()` returns the total number of descriptors which are ready and contained in the **`fd_set`** structures, 0 if the time limit has expired. Otherwise, it returns -1, and the global variable **`errno`** will contain one of the following values.

Error Codes

`EINVAL`	**`readfds`**, **`writefds`** and **`exceptfds`** are all NULL
`EBADF`	one of the descriptor sets contains an entry which is not a socket

send	send data on a connected socket	**Data Input/Output**

Syntax

```
#include  <sdksock.h>
int  send ( int s, const char *buf, int len, int flags );
```

Arguments

`s`	a descriptor identifying a connected socket
`buf`	a buffer containing the data to be transmitted
`len`	the length of the data in **`buf`**
`flags`	specifies the way in which the call is made

Description

`send()` is used on connected datagram or stream sockets and is used to write outgoing data on a socket. For datagram sockets, care must be taken not to exceed the maximum IP packet size of the underlying subnets.

Note that the successful completion of a **`send()`** does not indicate that the data was successfully delivered.

If no buffer space is available within the transport system to hold the data to be transmitted, **`send()`** will block unless the socket has been placed in a non-blocking I/O mode. On non-blocking **`SOCK_STREAM`** sockets, the number of bytes written may be between 1 and the requested length, depending on buffer availability on both the local and foreign hosts. The **`select()`** call may be used to determine when it is possible to send more data.

Flags may be used to influence the behavior of the function invocation beyond the options specified for the associated socket. That is, the semantics of this function are determined by the socket options and the flags parameter. The latter is constructed by "or-ing" any of the following values:

Value	**Meaning**
`MSG_OOB`	send out-of-band data

Return Value

If there are no errors, **`send()`** returns the total number of characters sent (note that this may be less than the number indicated by **`len`**.). Otherwise, it returns -1, and the global variable **`errno`** will contain one of the following values.

Error Codes

`EFAULT`	**`buf`** argument is not in a valid part of the user address space
`ENOTCONN`	socket is not connected
`EBADF`	descriptor is not a socket
`EOPNOTSUPP`	**`MSG_OOB`** was specified, but the socket is not of type **`SOCK_STREAM`**
`ESHUTDOWN`	socket has been shut down; it is not possible to **`send()`** on a socket after **`shutdown()`** has been invoked with **`how`** set to 1 or 2
`EWOULDBLOCK`	socket is marked as non-blocking and the requested operation would block
`EFBIG`	data written exceeds system capacity

sendto	send data to a specific destination	**Data Input/Output**

Syntax

```
#include  <sdksock.h>
int  sendto ( int s, char *buf, int len, int flags, SOCKADDR *to, int
tolen );
```

Arguments

s	a descriptor identifying a socket
buf	a buffer containing the data to be transmitted
len	length of the data in **buf**.
flags	specifies the way in which the call is made
to	an optional pointer to the address of the target socket
tolen	the size of the address in **to**

Description

sendto() is used on datagram or stream sockets and is used to write outgoing data on a socket. Note that the successful completion of a **sendto()** does not indicate that the data was successfully delivered.

sendto() is normally used on a **SOCK_DGRAM** socket to send a datagram to a specific peer socket identified by the to parameter. On a **SOCK_STREAM** socket, the **to** and **tolen** parameters are ignored; in this case the **sendto()** is equivalent to **send()**.

To send a broadcast (on a **SOCK_DGRAM** only), the address in the **to** parameter should be constructed using the special IP address **INADDR_BROADCAST** (defined in **sdksock.h**) together with the intended port number. It is generally inadvisable for a broadcast datagram to exceed the size at which fragmentation may occur, which implies that the data portion of the datagram (excluding headers) should not exceed 512 bytes.

If no buffer space is available within the transport system to hold the data to be transmitted, **sendto()** will block unless the socket has been placed in a non-blocking I/O mode. On non-blocking **SOCK_STREAM** sockets, the number of bytes written may be between 1 and the requested length, depending on buffer availability on both the local and foreign hosts. The **select()** call may be used to determine when it is possible to send more data.

flags may be used to influence the behavior of the function invocation beyond the options specified for the associated socket. That is, the semantics of this function is determined by the socket options and the **flags** parameter. The latter is constructed by “or-ing” any of the following values:

Value	Meaning
MSG_OOB	send out-of-band data (**SOCK_STREAM** only)

Return Value

If there are no errors, **sendto()** returns the total number of characters sent (note that this may be less than the number indicated by **len**). Otherwise, it returns -1, and the global variable **errno** will contain one of the following values.

`sendto`	send data to a specific destination	**Data Input/Output**

Error Codes

`EFAULT`	**`buf`** or **`to`** parameters are not part of the user address space, or the **`to`** argument is too small (less than the size of a **`SOCKADDR`**)
`ENOBUFS`	system had insufficient resources to perform the operation
`ENOTCONN`	socket is not connected (**`SOCK_STREAM`** only)
`EBADF`	descriptor is not a socket
`EOPNOTSUPP`	**`MSG_OOB`** was specified, but the socket is not of type **`SOCK_STREAM`**
`ESHUTDOWN`	socket has been shutdown; it is not possible to **`sendto()`** on a socket after **`shutdown()`** has been invoked with **`how`** set to 1 or 2
`EWOULDBLOCK`	socket is marked as non-blocking and the requested operation would block
`EINVAL`	socket has not been bound with **`bind()`**

setsockopt	set a socket option	**Socket Control**

Syntax

```
#include  <sdksock.h>
int  setsockopt ( int s, int level, int optname, char *optval, int
optlen );
```

Arguments

s	a descriptor identifying a socket
level	the level at which the option is defined; the only supported level is **SOL_SOCKET**
optname	the socket option for which the value is to be set
optval	a pointer to the buffer in which the value for the requested option is supplied
optlen	the size of the **optval** buffer

Description

setsockopt() sets the current value for a socket option associated with a socket of any type, in any state. Although options may exist at multiple protocol levels, this specification only defines options that exist at the uppermost "socket'' level. Options affect socket operations, such as whether keep-connection message is sent in the normal data stream, whether **closesocket()** operation is graceful, etc.

There are two types of socket options: Boolean options that enable or disable a feature or behavior, and options which require an integer value or structure. To enable a Boolean option, **optval** points to a nonzero integer. To disable the option, **optval** points to an integer equal to zero. **optlen** should be equal to **sizeof(int)** for Boolean options. For other options, **optval** points to a structure that contains the desired value for the option, and **optlen** is the length of the structure.

SO_LINGER controls the action taken when unsent data is queued on a socket and a **closesocket()** is performed. See **closesocket()** for a description of the way in which the **SO_LINGER** settings affect the semantics of **closesocket()**. The application sets the desired behavior by creating a struct linger (pointed to by the **optval** argument) with the following elements:

```
struct linger {
    int    l_onoff;
    int    l_linger;
}
```

To enable **SO_LINGER**, the application should set **l_onoff** to a non-zero value, set **l_linger** to 0 or the desired timeout (seconds), and call **setsockopt()**. The timeout value should be within 0 and 10 (seconds). To enable **SO_DONTLINGER** (i.e., disable **SO_LINGER**) **l_onoff** should be set to zero and **setsockopt()** should be called.

An application may request that the implementation enable the use of "keep-alive" packets on TCP connections by turning on the **SO_KEEPALIVE** socket option. The precise semantics are implementation-specific, but should conform to section 4.2.3.6 of RFC 1122.

setsockopt	set a socket option	**Socket Control**

The following options are supported for **`setsockopt()`**. The Type identifies the type of data addressed by **`optval`**.

Value	Type	Meaning
`SO_DONTLINGER`	BOOL	do not block close waiting for unsent data to be sent; setting this option is equivalent to setting **`SO_LINGER`** with **`l_onoff`** set to zero
`SO_KEEPALIVE`	BOOL	send keepalives
`SO_LINGER`	LINGER	linger on close if unsent data is present

Return Value

If there are no errors, **`setsockopt()`** returns 0. Otherwise, it returns -1, and the global variable **`errno`** will contain one of the following values.

Error Codes

`EFAULT`	**`optval`** is not in a valid part of the process address space
`EINVAL`	**`level`** is not valid, or the information in **`optval`** is not valid
`ENOPROTOOPT`	the option is unknown or unsupported
`EBADF`	the descriptor is not a socket

shutdown	disable sends and/or receives on a socket	**Socket Control**

Syntax

```
#include  <sdksock.h>
int  shutdown ( int s, int how );
```

Arguments

`s`	a descriptor identifying a socket
`how`	a flag that describes what types of operation will no longer be allowed

Description

`shutdown()` is used on all types of sockets to disable reception, transmission, or both.

If **`how`** is 0, subsequent receives on the socket will be disallowed. This has no effect on the lower protocol layers. For TCP, the TCP window is not changed and incoming data will be accepted (but not acknowledged) until the window is exhausted. For UDP, incoming datagrams are accepted and queued.

If **`how`** is 1, subsequent sends are disallowed. For TCP sockets, an FIN will be sent. Setting **`how`** to 2 disables both sends and receives as described above.

Note that **`shutdown()`** does not close the socket, and resources attached to the socket will not be freed until **`closesocket()`** is invoked.

Return Value

If there are no errors, **`shutdown()`** returns 0. Otherwise, it returns -1, and the global variable **`errno`** will contain one of the following values.

Error Codes

`EINVAL`	**`how`** is not valid
`ENOTCONN`	socket is not connected (**`SOCK_STREAM`** only)
`EBADF`	descriptor is not a socket

`socket`	create a socket	**Socket Control**

Syntax

```
#include  <sdksock.h>
int socket ( int af, int type, int protocol );
```

Arguments

`af`	an address format specification; the only format currently supported is **`AF_INET`**, which is the ARPA Internet address format
`type`	a type specification for the new socket
`protocol`	a particular protocol to be used with the socket, or 0 if the caller does not wish to specify a protocol

Description

`socket()` allocates a socket descriptor of the specified address family, data type and protocol, as well as related resources. If a protocol is not specified (i.e., equal to 0), the default for the specified connection mode is used.

Only a single protocol exists to support a particular socket type using a given address format. The protocol number to use is particular to the "communication domain" in which communication is to take place.

The following type specifications are supported:

`SOCK_STREAM` — provides sequenced, reliable, two-way, connection-based byte streams with an out-of-band data transmission mechanism; uses TCP for the Internet address family

`SOCK_DGRAM` — supports datagrams, which are connectionless, unreliable buffers of a fixed (typically small) maximum length; uses UDP for the Internet address family

Sockets of type **`SOCK_STREAM`** are full-duplex byte streams. A stream socket must be in a connected state before any data may be sent or received on it. A connection to another socket is created with a **`connect()`** call. Once connected, data may be transferred using **`send()`** and **`recv()`** calls. When a session has been completed, a **`closesocket()`** must be performed. These communication protocols have been developed for **`SOCK_STREAM`** to ensure that data is not lost or duplicated.

`SOCK_DGRAM` sockets allow sending and receiving of datagrams to and from arbitrary peers using **`sendto()`** and **`recvfrom()`**. If a connection is established between the socket and a specific peer using **`connect()`**, datagrams may be sent to that peer using **`send()`** and may be received from (only) this peer using **`recv()`**.

Return Value

If there are no errors, **`socket()`** returns a descriptor referencing the new socket. Otherwise, it returns -1, and the global variable **`errno`** will contain one of the following values.

Error Codes

`EMFILE`	no more file descriptors are available
`EPROTONOSUPPORT`	the specified address family or protocol is not supported

Simplified Socket Library Reference

net_get_gateway	get local default gateway	**Port Inquiry**

Syntax

```
#include  <sdksys.h>
u_long  net_get_gateway ( void );
```

Arguments

N/A

Description

get local default gateway

Return Value

default gateway IP address

net_get_IP	get local IP address	**Port Inquiry**

Syntax

```
#include  <sdksys.h>
u_long  net_get_IP ( void );
```

Arguments

N/A

Description

get local IP address

Return Value

local IP address

net_get_MAC_address	get MAC address	**Port Inquiry**

Syntax

```
#include  <sdksys.h>
void   net_get_MAC_address ( u_char *mac );
```

Arguments

mac	get MAC address buffer pointer; this buffer must be 6-byte length

Description

get MAC address

Return Value

system copies the host MAC address to the mac input buffer

`net_get_netmask`	get local subnet mask	**Port Inquiry**

Syntax
```
#include  <sdksys.h>
u_long  net_get_netmask ( void );
```

Arguments

N/A

Description

get local subnet mask

Return Value

local netmask

`tcp_close`	close a local TCP port	**Socket Control**

Syntax
```
#include  <sdknet.h>
int tcp_close ( int handle );
```

Arguments

`handle`	the value returned from **`tcp_open()`**

Description

close a local TCP port

Return Value

`0`	OK
`-1`	error handle number

`tcp_connect`	connect to specific host IP and port	**Socket Control**

Syntax
```
#include  <sdknet.h>
int tcp_connect ( int handle, u_long rip, int rport, long tout );
```

Arguments

`handle`	the value return from **`tcp_open()`**
`rip`	remote host IP address that user wants to link
`rport`	remote host TCP port number
`tout`	wait for TCP connection time out value (milliseconds); 0 will wait for OK or fail

Description

connect to specific host IP and port

Return Value

`1`	connect OK
`0`	connect fail
`-1`	error handle number
`-2`	this handle is not a TCP handle
`-3`	timeout counter reached
`-4`	error state; already connected
`-5`	the **`rip:rport`** already in use

tcp_connect_nowait	connect to specific host IP and port no wait	**Socket Control**

Syntax

```
#include  <sdknet.h>
int  tcp_connect_nowait ( int handle, u_long rip, int rport );
```

Arguments

handle	the value returned from **tcp_open()**
rip	remote host IP address that user wants to link to
rport	remote host's TCP port number

Description

connect to specific host's IP and port without waiting

Return Value

0	start to connect
-1	error handle number
-2	error argument
-3	error state; already connected
-4	the **rip:rport** is already in use

tcp_get_remote	get connected host's IP and port	**Socket Inquiry**

Syntax

```
#include  <sdknet.h>
int  tcp_get_remote ( int handle, u_long *rip, int *rport )
```

Arguments

handle	the value returned from **tcp_open()**
rip	connected host's IP address pointer
rport	connected host's TCP port number pointer

Description

get connected host's IP and port

Return Value

0	get ok
-1	error handle
-2	error argument
-3	no connection

`tcp_iqueue`	get the length of data accumulated in TCP driver's input buffer	**Socket Inquiry**

Syntax

```
#include  <sdknet.h>
int  tcp_iqueue ( int handle );
```

Arguments

`handle`	the value returned from **`tcp_open()`**

Description

get the length of data accumulated in TCP driver's input buffer

Return Value

`>=0`	TCP input buffer queued data size
`-1`	error handle number
`-2`	this is not a TCP handle
`-3`	TCP not connected

`tcp_listen`	place a socket in a state where it is listening for an incoming connection	**Socket Control**

Syntax

```
#include  <sdknet.h>
int  tcp_listen ( int handle, long tout );
```

Arguments

`handle`	the return value from **`tcp_open()`**
`tout`	wait for listen time out value (milliseconds); 0 will wait for someone to connect

Description

places a socket a state where it is listening for an incoming connection

Return Value

`1`	connect OK or already connected
`0`	connect fail
`-1`	error handle number
`-2`	this handle is not a TCP handle
`-3`	timeout counter reached
`-4`	error state; already connected

tcp_listen_nowait	place a socket in a state where it is listening for an incoming connection without waiting	**Socket Control**

Syntax

```
#include  <sdknet.h>
int  tcp_listen_nowait ( int handle );
```

Arguments

handle	the value returned from **tcp_open()**

Description

place a socket a state where it is listening for an incoming connection without waiting

Return Value

0	start to listen
-1	error handle number
-2	this handle is not a TCP handle
-3	error state; already connected

tcp_listento	listen for a specific incoming connection	**Socket Control**

Syntax

```
#include  <sdknet.h>
int  tcp_listento ( int handle, u_long rip, int rport, long tout );
```

Arguments

handle	the value returned from **tcp_open()**
rip	remote host IP address that user wants to link to; 0 indicates any remote IP address
rport	remote host TCP port number; 0 indicates any TCP port number
tout	wait for listen timeout value (milliseconds)

Description

listen for a specific incoming connection

Return Value

1	connect OK or already connected
0	connect fail
-1	error handle number
-2	this handle is not a TCP handle
-3	timeout counter reached
-4	error state; already connected

tcp_listento_nowait	listen for a specific incoming connection without waiting	**Socket Control**

Syntax
```
#include  <sdknet.h>
int  tcp_listento_nowait ( int handle, u_long rip, int rport );
```

Arguments

handle	the value returned from **tcp_open()**
rip	remote host IP address that user wants to link to; 0 indicates any remote IP address
rport	remote host's TCP port number; 0 indicates any TCP port number

Description

listen for a specific incoming connection no wait

Return Value

0	start to listen
-1	error handle number
-2	this handle is not a TCP handle
-3	error state; already connected

tcp_ofree	get amount of free space in TCP driver's input buffer	**Socket Inquiry**

Syntax
```
#include  <sdknet.h>
int  tcp_ofree ( int handle );
```

Arguments

handle	the value returned from **tcp_open()**

Description

get amount of free space in TCP driver's input buffer

Return Value

>=0	TCP output buffer's free size
-1	error handle number
-2	this is not a TCP handle
-3	TCP not connected

tcp_open	open a local TCP port	**Socket Control**

Syntax
```
#include  <sdknet.h>
int  tcp_open ( int port );
```

Arguments

port	local TCP port number

Description

open a local TCP port

Return Value

>=0	open handle
-1	open fail

`tcp_recv`	receive data from a connected socket	**Data Input/Output**

Syntax

```
#include  <sdknet.h>
int  tcp_recv ( int handle, char *buffer, int len );
```

Arguments

`handle`	the value returned from **`tcp_open()`**
`buffer`	the pointer to buffer for incoming data
`len`	the size of **`buffer`** (bytes)

Description

receives data from a connected socket

Return Value

`>=0`	received data length
`-1`	error handle number
`-2`	error argument
`-3`	TCP not connected

`tcp_send`	sends data to a connected socket	**Data Input/Output**

Syntax

```
#include  <sdknet.h>
int  tcp_send ( int handle, char *buffer, int len );
```

Arguments

`handle`	the value return from **`tcp_open()`**
`buffer`	the pointer to buffer for outgoing data
`len`	the length of data in buffer to be sent (bytes)

Description

sends data on a connected socket

Return Value

`>=0`	outgoing data length
`-1`	error handle number
`-2`	error argument
`-3`	TCP not connected

tcp_state	get TCP state	**Socket Inquiry**

Syntax

```
#include  <sdknet.h>
int  tcp_state ( int handle );
```

Arguments

handle	the value returned from **tcp_open()**

Description

get TCP state

Return Value

0	TCP closed
1	TCP listen
2	TCP connecting
3	TCP connected
4	TCP close wait (remote closed)
5	TCP closing
-1	error handle
-2	this handle is not a TCP handle

udp_close	close a local UDP port	**Socket Control**

Syntax

```
#include  <sdknet.h>
int  udp_close ( int handle );
```

Arguments

handle	the value return from **udp_open()**

Description

close a local UDP port

Return Value

0	close OK
-1	error handle number

udp_iqueue	get length of data accumulated in UDP driver's input buffer	**Socket Inquiry**

Syntax

```
#include  <sdknet.h>
int  udp_iqueue ( int handle )
```

Arguments

handle	the value returned from **udp_open()**

Description

get the length of data accumulated in UDP driver's input buffer

Return Value

>=0	UDP input buffer queued data size
-1	error handle number
-2	this is not a UDP handle

`udp_ofree`	get amount of free space in UDP driver's input buffer	**Socket Inquiry**

Syntax

```
#include  <sdknet.h>
int  udp_ofree ( int handle );
```

Arguments

`handle`	the value returned from **`udp_open()`**

Description

amount of free space in UDP driver's input buffer

Return Value

`>=0`	UDP output buffer free size
`-1`	error handle number
`-2`	this is not a UDP handle

`udp_open`	open a local UDP port	**Socket Inquiry**

Syntax

```
#include  <sdknet.h>
int  udp_open ( int port );
```

Arguments

`port`	local UDP port number

Description

open a local UDP port

Return Value

`>=0`	open handle
`-1`	open fail

`udp_recv`	receive data from a specific source address	**Data Input/Output**

Syntax

```
#include  <sdknet.h>
int udp_recv ( int handle, u_long *rip, int *rport, char *buf, int len );
```

Arguments

`handle`	the value returned from **`udp_open()`**
`rip`	the remote host's IP address
`rport`	pointer to the remote UDP port number
`buf`	pointer to buffer for incoming data
`len`	the length of **`buf`** (bytes)

Description

receive data from a specific source address

Return Value

`>= 0`	length of data received
`-1`	receive failed

udp_send	sends data to a specific destination	**Data Input/Output**

Syntax

```
#include  <sdknet.h>
int  udp_send ( int handle, u_long rip, int rport, char *buf, int len );
```

Arguments

handle	the value returned from **udp_open()**
rip	destination host IP address
rport	destination host UDP port number
buf	pointer to buffer for outgoing data
len	send data length (bytes)

Description

send data to a specific destination

Return Value

>= 0	length of data sent out
-1	send failed

System Control Library Reference

sys_calloc	allocates an array in memory with elements initialized to 0	

Syntax

```
#include  <sdksys.h>
void  *sys_calloc ( unsigned nelem, unsigned elsize );
```

Arguments

nelem	number of elements to be allocated
elsize	length of each element (bytes)

Description

The **sys_calloc()** function allocates space for **nelem** objects, each **elsize** bytes in length. The result is identical to calling **sys_malloc()** with an argument of '**nelem * elsize**', with the exception that the allocated memory is explicitly initialized to zero bytes.

Return Value

The **sys_calloc()** function returns a pointer to the allocated memory if successful; otherwise a NULL pointer is returned.

sys_clock_ms	read the time count (milliseconds) from power-up	

Syntax

```
#include  <sdksys.h>
unsigned long  sys_clock_ms ( void );
```

Arguments

N/A

Description

read the NE-4100-P's time count (milliseconds) from power-up

Return Value

This function returns the time counter in milliseconds.

`sys_clock_s`	read the time count (seconds) from power-up	

Syntax

```
#include  <sdksys.h>
unsigned long  sys_clock_s ( void );
```

Arguments

N/A

Description

read the NE-4100-P's time count (seconds) from power-up

Return Value

This function returns the time counter in seconds.

`sys_exit`	exit application	

Syntax

```
#include  <sdksys.h>
void  sys_exit ( void );
```

Arguments

N/A

Description

exit user application and return to kernel; it will stop the user application

Return Value

N/A

`sys_free`	deallocate or free a memory block	

Syntax

```
#include  <sdksys.h>
void  sys_free ( void *ptr );
```

Arguments

`ptr`	pointer to a memory returned by **`sys_malloc()`** or **`sys_calloc()`**

Description

The **`sys_free()`** function causes the allocated memory referenced by **`ptr`** to be made available for future allocations.

Return Value

N/A

sys_get_info	get general server information	

Syntax

```
#include  <sdksys.h>
int  sys_get_info ( struct sdk_sysinfo *info );
```

Arguments

info	a pointer to a buffer to receive the general server information

Description

General server information for the NE-4100-P is returned with the following structure:

```
struct sdk_sysinfo {
    struct sdk_version firmware_version;    /* Server’s firmware
    version. */
    unsigned long      serial_no;      /* Server’s serial number */
    unsigned short     product_id;     /* Server’s product ID */
    unsigned char      MAC_addr[6];    /* Server Ethernet MAC address
    */
    struct sdk_version ap_version;     /* User’s AP version */
    unsigned short     ap_date_year;   /* Date of AP: A.D. e.g. 2002 */
    unsigned char      ap_date_month;  /* Range: 1 - 12 */
    unsigned char      ap_date_day;    /* Range: 1 - 31 */
    unsigned char      ap_time_hour;   /* Range: 0 - 23 */
    unsigned char      ap_time_minute;/* Range: 0 - 59 */
};
```

The version information is stored with the following structure:

```
struct sdk_version {
    unsigned short     ext_version;
    unsigned char      sub_version;
    unsigned char      main_version;
};
```

For version 1.20.3, the **main_version** is 1, **sub_version** is 20 and **ext_version** is 3.

Return Value

sys_get_info() returns the length of the information structure. A return value of 0 indicates the argument is invalid.

sys_get_LastErrno	get last error number related to a socket	

Syntax

```
#include  <sdksys.h>
int  sys_get_LastErrno ( int socket, int *err );
```

Arguments

socket	a descriptor identifying a socket
err	a pointer to where to place the last error number

Description

Most sockets APIs will put the error reason as the error number in the global variable **errno**. In some multi-threading application, the variable may be overwritten suddenly by another thread. In such case, you can use this function to retrieve the last error encountered by the specified socket. On return, the error number is placed in the space specified by **err**.

Note that the returned error number is meaningful only when the last socket operation on specified socket failed. Otherwise, the returned error number is undefined.

Return Value

-3	the pointer is invalid
0	OK

sys_get_SerialType	get async port interface signal type	

Syntax

```
#include  <sdksys.h>
int  sys_get_SerialType ( int port );
```

Arguments

port	async serial port number

Description

get async port interface signal type

Return Value

0	RS-23
1	RS-42
2	RS-485 2-wire
3	RS-485 4-wire
-1	bad port

`sys_getFreeMemSize`	get available memory status used by **`sys_malloc()`** and **`sys_calloc()`**	

Syntax

```
#include  <sdksys.h>
int  sys_getFreeMemSize ( long *total_size, long *max_block_size );
```

Arguments

`total_size`	pointer to buffer to retrieve total size of free memory (bytes)
`max_block_size`	pointer to buffer to retrieve the maximum size of free memory blocks

Description

When a program requests to allocate memory with **`sys_malloc()`** or so on, the system will return a block of continuous free memory. After allocation and deallocation occurs several times, the entire memory space may contain several small blocks of free memory of different sizes. Although the total free memory may still be large enough, **`sys_malloc()`** may fail if there is no contiguous block large enough for the desired size.

After **`sys_getFreeMemorySize()`** is called, the total size of free memory is put in the buffer pointed by **`total_size`**, and the maximum size of free memory blocks is put in the buffer pointed by **`max_block_size`**.

This function is useful for debugging and may be used to determine if there is a memory leak.

Return Value

`0`	This function always returns 0.

`sys_GetServersIp`	retrieve the DNS server's and time server's address supplied by DHCP/BOOTP server	

Syntax

```
#include  <sdksys.h>
int  sys_GetServersIp ( unsigned long *dns1_ip, unsigned long *dns2_ip,
unsigned long *time_ip );
```

Arguments

`dns1_ip`	pointer to buffer to retrieve IP address of first DNS server; if **`dns1_ip`** is NULL, this address is not returned
`dns2_ip`	pointer to buffer to retrieve IP address of second DNS server; if **`dns2_ip`** is NULL, this address is not returned
`time_ip`	pointer to buffer to retrieve IP address of time server; if **`time_ip`** is NULL, time server's IP is not returned

Description

The NE-4100-P's network configuration can be set to DHCP or BOOTP mode by **`sysc_SetIPConfig()`**. In either of these modes, the DHCP/BOOTP server may assign IP addresses for the DNS server and time server. **`sys_GetServersIp()`** returns this information to the user program using the specified buffers.

If the requested IP address is not provided, the value of 0 will be returned in the corresponding buffer.

Return Value

`0`	This function always returns 0.

`sys_malloc`	allocates memory blocks	

Syntax

```
#include  <sdksys.h>
void  *sys_malloc ( unsigned size );
```

Arguments

`size`	number of elements to be allocated

Description

The **`sys_malloc()`** function allocates **`size`** bytes of memory. The allocated space is suitably aligned (after possible pointer coercion) for storage of any type of object.

Note that **`sys_malloc()`** does NOT normally initialize the returned memory to zero bytes.

Return Value

The **`sys_malloc()`** function returns a pointer to the allocated memory if successful; otherwise a NULL pointer is returned.

`sys_realloc`	reallocate memory blocks	

Syntax

```
#include  <sdksys.h>
void  *sys_realloc ( void *ptr, unsigned newsize );
```

Arguments

`ptr`	pointer to previously allocated memory block
`newsize`	new size (bytes)

Description

The **`sys_realloc()`** function changes the size of the previously allocated memory referenced by **`ptr`** to **`newsize`** bytes. The contents of the memory are unchanged up to the lesser of the new and old sizes. If the new size is larger, the value of the newly allocated portion of the memory is undefined. If the requested memory cannot be allocated, NULL is returned and the memory referenced by **`ptr`** is valid and unchanged. If **`ptr`** is NULL, the **`sys_realloc()`** function behaves identically to **`sys_malloc()`** for the specified size.

Return Value

The **`sys_realloc()`** function returns a pointer, possibly identical to **`ptr`**, to the allocated memory if successful; otherwise a NULL pointer is returned. The **`sys_realloc()`** function always leaves the original buffer intact if an error occurs.

`sys_restart_system`	restart system	

Syntax

```
#include  <sdksys.h>
void   sys_restart_system ( void );
```

Arguments

N/A

Description

restart system

Return Value

N/A

`sys_restart_UserAP`	restart user AP	

Syntax
```
#include  <sdksys.h>
void  sys_restart_UserAP ( void );
```
Arguments

N/A

Description

restart user AP

Return Value

N/A

`sys_Set_RegisterID`	set application ID	

Syntax
```
#include  <sdksys.h>
void  sys_Set_RegisterID ( u_long id );
```
Arguments

id	application ID; 0x00000000 to 0x7FFFFFFF only

Description

set the application ID; IDs between 0x80000000 and 0xFFFFFFFF are reserved for MOXA only; this function should be called as soon as application runs

Return Value

N/A

`sys_set_SerialType`	set async port interface signal type	

Syntax
```
#include  <sdksys.h>
int  sys_set_SerialType ( int port, int type );
```
Arguments

port	async serial port number
type	0: RS-232 1: RS-422 2: RS-485 2-wire 3: RS-485 4-wire

Description

set async port interface signal type

Return Value

0	set OK
-1	bad port
-2	bad parameter (cannot set this interface type)

sys_sleep_ms	sleep task (milliseconds)	

Syntax

```
#include  <sdksys.h>
int  sys_sleep_ms ( long time_ms );
```

Arguments

time_ms	sleep time (milliseconds)

Description

sleep task (milliseconds)

Return Value

This function always returns **0**.

sys_timeout	set the timeout event service routine	

Syntax

```
#include  <sdksys.h>
int  sys_timeout ( void (*func)(), long time_ms );
```

Arguments

func	the timeout event service routine
time_ms	timeout value (milliseconds)

Description

set the timeout event service routine

Return Value

0	no errors
EINVAL	the **isr** argument event function pointer is invalid
ENOBUFS	no resources

sysc_GetDebug	get debug output setting	

Syntax

```
#include  <sdkconf.h>
int  sysc_GetDebug ( void );
```

Arguments

N/A

Description

get debug output setting

Return Value

0	debug mode off
1	debug mode on

`sysc_GetGateway`	get IP address of gateway	

Syntax
```
#include  <sdkconf.h>
u_long  sysc_GetGateway ( void );
```
Arguments

N/A

Description

get server gateway

Return Value

gateway IP address

`sysc_GetIP`	get IP address	

Syntax
```
#include  <sdkconf.h>
u_long  sysc_GetIP ( void );
```
Arguments

N/A

Description

get server IP address

Return Value

the local server IP address

`sysc_GetIPConfig`	get IP configuration settings	

Syntax
```
#include  <sdkconf.h>
int  sysc_GetIPConfig ( void );
```
Arguments

N/A

Description

get the IP configuration settings

Return Value

0	static IP
1	DHCP
2	DHCP & BOOTP
3	BOOTP

sysc_GetIPLocating	get IP location setting	

Syntax

```
#include  <sdkconf.h>
int  sysc_GetIPLocating ( u_long *ipaddr, u_int *pno, int *time );
```

Arguments

ipaddr	remote server IP address buffer pointer
pno	remote UDP port number pointer
time	report period time pointer

Description

get the NE-4100-P’s IP Location setting

Return Value

0	OK
-2	error argument

sysc_GetName	get server name	

Syntax

```
#include  <sdkconf.h>
int  sysc_GetName ( char *name, int size );
```

Arguments

name	server name buffer pointer
size	buffer size

Description

get server name

Return Value

>=0	the length of server name returned

sysc_GetNetmask	get netmask	

Syntax

```
#include  <sdkconf.h>
u_long  sysc_GetNetmask ( void );
```

Arguments

N/A

Description

get server netmask

Return Value

the local server IP netmask

`sysc_GetPassword`	get server password	

Syntax

```
#include  <sdkconf.h>
int  sysc_GetPassword ( char *password, int size );
```

Arguments

`password`	server password buffer pointer
`size`	buffer size

Description

get NE-4100-P password

Return Value

`>=0`	the length of password returned

`sysc_GetSerialFIFO`	get serial port FIFO settings	

Syntax

```
#include  <sdkconf.h>
int  sysc_GetSerialFIFO ( int port );
```

Arguments

`port`	async serial port number

Description

get the serial port FIFO settings

Return Value

`1`	FIFO enabled
`0`	FIFO disabled
`-1`	error port number

`sysc_GetSerialInterface`	get serial port interface	

Syntax

```
#include  <sdkconf.h>
int  sysc_GetSerialInterface ( int port );
```

Arguments

`port`	async serial port number

Description

get the serial port interface

Return Value

`-1`	error port number
`0`	RS-232
`1`	RS-422
`2`	RS-485 2-wire
`3`	RS-485 4-wire

sysc_GetSerialIoctl	get serial port parameters	

Syntax

```
#include  <sdkconf.h>
int  sysc_GetSerialIoctl ( int port, int *baud, int *mode, int *flow );
```

Arguments

port	async serial port number		
baud	baud rate buffer pointer		
	0: 50	6: 600	12: 9600
	1: 75	7: 1200	13: 19200
	2: 110	8: 1800	14: 38400
	3: 134.5	9: 2400	15: 57600
	4: 150	10: 4800	16: 115200
	5: 300	11: 7200	17: 230400
mode	character mode buffer pointer		
	bit_cnt (bits 0-1)		
	0x00: data bit 5		
	0x01: data bit 6		
	0x02: data bit 7		
	0x03: data bit 8		
	stop_bit (bit 2)		
	0x00: stop bit 1		
	0x04: stop bit 1.5 or 2		
	parity (bits 3,4 5)		
	0x00: none parity		
	0x08: odd parity		
	0x18: even parity		
	0x28: mark parity		
	0x38: space parity		
flow	flow control buffer pointer		
	0: none		
	1: RTS/CTS		
	2: XON/XOFF		
	3: DTR/DSR		

Description

get serial port parameter

Return Value

0	OK
-1	error port number

`sysc_SaveAndRestart`	save new settings and restart NE-4100-P	

Syntax

```
#include  <sdkconf.h>
void  sysc_SaveAndRestart ( void );
```

Arguments

N/A

Description

save new settings and restart the NE-4100-P; after calling this function, the NE-4100-P will be restarted

Return Value

N/A

`sysc_SetDebug`	set debug output setting	

Syntax

```
#include  <sdkconf.h>
int  sysc_SetDebug ( int mode );
```

Arguments

mode	0: off 1: on

Description

set debug output setting

Return Value

0	OK
-2	error argument

`sysc_SetGateway`	set gateway address	

Syntax

```
#include  <sdkconf.h>
int  sysc_SetGateway ( u_long ipaddr );
```

Arguments

ipaddr	new gateway IP address

Description

set gateway

Return Value

0	OK
-2	error argument

`sysc_SetIP`	set IP address	

Syntax

```
#include  <sdkconf.h>
int  sysc_SetIP ( u_long ipaddr );
```

Arguments

`ipaddr`	new local server IP address

Description

set IP address

Return Value

`0`	OK
`-2`	error argument

`sysc_SetIPConfig`	define how IP address, netmask and gateway are obtained	

Syntax

```
#include  <sdkconf.h>
int  sysc_SetIPConfig ( int type );
```

Arguments

`type`	IP configuration 0: static IP 1: DHCP 2: DHCP & BOOTP 3: BOOTP

Description

define how IP address, netmask and gateway are obtained

Return Value

`0`	OK
`-2`	error argument

`sysc_SetIPLocating`	set IP location function	

Syntax

```
#include  <sdkconf.h>
int  sysc_SetIPLocating ( u_long ipaddr, int pno, int time );
```

Arguments

`ipaddr`	IP address of IP location remote server; set 0.0.0.0 to disable this function
`pno`	UDP port number of IP location remote server
`time`	report period time (seconds)

Description

set IP location function

Return Value

`0`	OK
`-2`	error argument

sysc_SetName	set server name	

Syntax

```
#include  <sdkconf.h>
int  sysc_SetName ( char *name );
```

Arguments

name	new server name

Description

set server name

Return Value

0	OK

sysc_SetNetmask	set netmask	

Syntax

```
#include  <sdkconf.h>
int  sysc_SetNetmask ( u_long netmask );
```

Arguments

netmask	new local server netmask

Description

set NE-4100-P netmask

Return Value

0	OK

sysc_SetPassword	set password	

Syntax

```
#include  <sdkconf.h>
int  sysc_SetPassword ( char *password );
```

Arguments

password	new server password

Description

set NE-4100-P password

Return Value

0	OK

sysc_SetSerialFIFO	set serial port FIFO settings	

Syntax

```
#include  <sdkconf.h>
int  sysc_SetSerialFIFO ( int port, int mode );
```

Arguments

port	async serial port number
mode	FIFO mode 0: disable 1: enable

Description

set serial port FIFO settings

Return Value

0	OK
-1	error port number
-2	error argument

sysc_SetSerialInterface	set the serial port interface	

Syntax

```
#include  <sdkconf.h>
int  sysc_SetSerialInterface ( int port, int type );
```

Arguments

port	async serial port number
type	serial interface type 0: RS-232 1: RS-422 2: RS-485 2-wire 3: RS-485 4-wire

Description

set serial port interface

Return Value

0	OK
-1	error port number
-2	error argument

sysc_SetSerialIoctl	set serial port parameter	

Syntax

```
#include  <sdkconf.h>
int  sysc_SetSerialIoctl ( int port, int baud, int mode, int flow );
```

Arguments

port	async serial port number		
baud	0: 50	6: 600	12: 9600
	1: 75	7: 1200	13: 19200
	2: 110	8: 1800	14: 38400
	3: 134.5	9: 2400	15: 57600
	4: 150	10: 4800	16: 115200
	5: 300	11: 7200	17: 230400

mode

bit_cnt OR **stop_bit** OR **parity**

bit_cnt (bits 0-1)
- 0x00: data bit 5
- 0x01: data bit 6
- 0x02: data bit 7
- 0x03: data bit 8

stop_bit (bit 2)
- 0x00: stop bit 1
- 0x04: stop bits 1.5 or 2

parity (bits 3- 5)
- 0x00: no parity
- 0x08: odd parity
- 0x18: even parity
- 0x28: mark parity
- 0x38: space parity

flow

flow control
- 0: none
- 1: RTS/CTS
- 2: XON/XOFF
- 3: DTR/DSR

Description

set serial port parameters

Return Value

0	OK
-1	error port number
-2	error argument

`sysc_SetToDefault`	set to default value	

Syntax

```
#include  <sdkconf.h>
void  sysc_SetToDefault ( void );
```

Arguments

N/A

Description

restore all configurations back to factory default value

Return Value

N/A

Flash ROM Access Library Reference

`flash_erase`	erase flash ROM	

Syntax

```
#include  <sdkflash.h>
int  flash_erase ( void );
```

Arguments

N/A

Description

erase flash ROM

Return Value

0	OK
-1	fail

`flash_length`	get current data length of flash ROM	

Syntax

```
#include  <sdkflash.h>
long  flash_length ( void );
```

Arguments

N/A

Description

get current data length of flash ROM

Return Value

>=0	current data length of flash ROM; max length is 163840 (160 KB)
0	value after calling **`sys_flash_erase()`**

`flash_read`	read data from flash ROM	

Syntax

```
#include  <sdkflash.h>
long  flash_read ( long offset, char *buffer, long size );
```

Arguments

offset	indicates the point at which the function starts reading the buffer, measured in bytes
buffer	read buffer pointer
size	buffer size

Description

read data from the flash ROM

Return Value

>=0	read data size
-1	read failed

`flash_write`	write data to flash ROM	

Syntax

```
#include  <sdkflash.h>
long  flash_write ( char *buffer, long size );
```

Arguments

buffer	write data buffer pointer
size	write data size, from 1 to 163840

Description

write data to the flash ROM; allows specific portions of flash ROM to be written

Return Value

>0	write length
-1	write failed
-2	need to erase flash ROM first

`sys_FlashErase`	erase flash ROM	

Syntax

```
#include  <sdkflash.h>
int  sys_FlashErase ( int bank );
```

Arguments

bank	memory bank, from 0 to 4

Description

erase flash ROM

Return Value

-1	fail
-2	argument error
0	OK

sys_FlashLength	get current data length of flash ROM	

Syntax

```
#include  <sdkflash.h>
long  sys_FlashLength ( int bank );
```

Arguments

bank	memory bank, from 0 to 4

Description

get current data length of flash ROM

Return Value

>=0	current data length of flash ROM; max length is 32768 (32KB)
0	value after calling **sys_FlashErase()**
-2	argument error

sys_FlashRead	read data to flash ROM	

Syntax

```
#include  <sdkflash.h>
long  sys_FlashRead ( int bank, long offset, char * buffer, long size );
```

Arguments

bank	memory bank, from 0 to 4
offset	start read offset from bank begin
buffer	read buffer pointer
size	write data size, from 1 to 32768

Description

read data to flash ROM; may be used to read specific segment of flash ROM rather than the entire flash ROM

Return Value

>=0	size of data read
-1	read fail
-2	argument error

sys_FlashWrite	write data to flash ROM	

Syntax

```
#include  <sdkflash.h>
long  sys_FlashWrite ( int bank, char *buffer, long size );
```

Arguments

bank	memory bank, from 0 to 4
buffer	write data buffer pointer
size	write data size, from 1 to 32768

Description

write data to flash ROM

Return Value

>0	length of data written
-1	write fail
-2	argument error
-3	need to erase flash ROM first

Debug Library Reference

Each of these functions returns the number of characters printed, or a negative value if an error occurs.

dbg_printf	print formatted output to debug output stream	

Syntax

```
#include  <sdkdbg.h>
int  dbg_printf ( char *format, [, argument]... );
```

Arguments

format	format control
argument	optional arguments

Description

print formatted output to debug output stream

Return Value

This function returns the number of characters printed, or a negative value if an error occurs.

dbg_put_block	print out a block of data for debugging	

Syntax

```
#include  <sdkdbg.h>
int  dbg_put_block ( char *buf, int len );
```

Arguments

buf	the print out debugging data buffer pointer
len	length of the debugging data buffer

Description

print out a block of data for debugging

Return Value

This function returns the length of data that is printed out.

dbg_put_ch	print out a character for debugging	

Syntax

```
#include  <sdkdbg.h>
int  dbg_put_ch ( char ch );
```

Arguments

ch	the character value that will be printed out

Description

print out a character for debugging

Return Value

This function returns the length of data that is printed out.

dbg_put_doubleword	print out a 4-byte unsigned long value for debugging	

Syntax

```
#include  <sdkdbg.h>
int  dbg_put_doubleword ( unsigned long value );
```

Arguments

value	the printed out unsigned long value

Description

print out a 4-byte unsigned long value for debugging

Return Value

This function returns the length of data that is printed out.

dbg_put_doubleword_hex	print out a 4-byte unsigned long value with HEX format for debugging	

Syntax

```
#include  <sdkdbg.h>
int  dbg_put_doubleword_hex ( unsigned long value );
```

Arguments

value	the printed out unsigned long value

Description

print out a 4-byte unsigned long value with HEX format for debugging

Return Value

This function returns the length of data that is printed out.

dbg_put_IP	print out an IP address in the a.b.c.d format for debugging	

Syntax

```
#include  <sdkdbg.h>
int  dbg_put_IP ( unsigned long ipaddr );
```

Arguments

ipaddr	the printed out Internet host’s IP address

Description

print out an IP address in the a.b.c.d format for debugging

Return Value

This function returns the length of data that is printed out.

dbg_put_string	print out a string for debugging	

Syntax

```
#include  <sdkdbg.h>
int  dbg_put_string ( char *buf );
```

Arguments

buf	the printed out debugging data buffer’s pointer

Description

print out a string for debugging

Return Value

This function returns the length of data that is printed out.

dbg_put_word	print out a 2-byte unsigned integer value for debugging	

Syntax

```
#include  <sdkdbg.h>
int  dbg_put_word ( unsigned short value );
```

Arguments

value	the printed out unsigned short value

Description

print out a 2-byte unsigned integer value for debugging

Return Value

This function returns the length of data that is printed out.

dbg_put_word_hex	print out a 2-byte unsigned integer value with HEX format for debugging	

Syntax

```
#include  <sdkdbg.h>
int  dbg_put_word_hex ( unsigned short value );
```

Arguments

value	the print out unsigned short value

Description

print out a 2-byte unsigned integer value with HEX format for debugging

Return Value

This function returns the length of data that is printed out.

DIO Library Reference

DIO_ControlSingleIO	set output channel to high or low state	

Syntax

```
#include  <sdkdio.h>
int  DIO_ControlSingleIO ( int io, int highlow );
```

Arguments

io	I/O number
highlow	0: set the output state to low != 0: set the output state to high

Description

set output channel to high or low state

Return Value

0	OK
-2	fail

DIO_GetSingleIO	get mode (input or output) of DIO channel	

Syntax

```
#include  <sdkdio.h>
int DIO_GetSingleIO ( int io, int *mode );
```

Arguments

io	I/O number
mode	pointer to buffer to retrieve the I/O mode *mode = 0 for input mode, 1 for output mode

Description

get mode (input or output) of DIO channel

Return Value

0	OK
-2	fail

DIO_GetSingleIOStatus	get current state of DIO channel (high or low)	

Syntax

```
#include  <sdkdio.h>
int  DIO_GetSingleIOStatus ( int io, int *highlow );
```

Arguments

io	I/O number
highlow	pointer to buffer to retrieve state of DIO channel *highlow = 0 for low state, 1 for high state

Description

get current state of DIO channel (high or low)

Return Value

0	OK
-2	fail

DIO_SetSingleIO	set DIO channel to input or output mode	

Syntax

```
#include  <sdkdio.h>
int  DIO_SetSingleIO ( int io, int mode );
```

Arguments

io	I/O number
mode	0: set to input !=0: set to output

Description

set DIO channel to input or output mode

Return Value

0	OK
-2	fail

Scf_getSDioMode	get initial mode (input or output) for DIO channel	

Syntax

```
#include  <sdkdio.h>
int Scf_getSDioMode ( int io, int *mode );
```

Arguments

io	I/O number
mode	pointer to buffer to retrieve the initial I/O mode *mode = 0 for input mode, 1 for output mode

Description

get initial I/O mode (input or output) for DIO channel

Return Value

0	OK
-2	fail

Scf_getSDioState	get initial output state (high or low) for output channel	

Syntax

```
#include  <sdkdio.h>
int  Scf_getSDioState ( int io, int *highlow );
```

Arguments

io	I/O number
highlow	pointer to buffer to retrieve initial I/O state *highlow = 0 for low state, 1 for high state

Description

get initial output state (high or low) for output channel; this setting takes effect only if the initial I/O mode is set to output mode.

Return Value

0	OK
-2	fail

`Scf_setSDioMode`	set initial mode of DIO channel to input or output	

Syntax

```
#include  <sdkdio.h>
int  Scf_setSDioMode ( int io, int mode );
```

Arguments

`io`	I/O number
`mode`	0: set initial mode to input != 0: set initial mode to output

Description

set the initial mode of DIO channel to input or output; this function defines the channel's initial mode when NE-4100-P boots up

Return Value

`0`	OK
`-2`	fail

`Scf_setSDioState`	set initial state of output channel to high or low	

Syntax

```
#include  <sdkdio.h>
int  Scf_setSDioState ( int io, int highlow );
```

Arguments

`io`	I/O number
`highlow`	0: set the initial output state to low != 0: set the initial output state to high

Description

set initial state of output channel to high or low

Return Value

`0`	OK
`-2`	fail

Thread Control Library Reference

`sys_ThreadClose`	close a thread	

Syntax

```
#include  <sdktask.h>
int  sys_ThreadClose ( unsigned long threadID );
```

Arguments

`threadID`	thread ID returned from **`sys_ThreadCreate()`**

Description

close a thread

Return Value

`0`	OK
`-1`	error threadID

sys_ThreadCreate	create a thread	

Syntax

```
#include  <sdktask.h>
int  sys_ThreadCreate ( void (*ap) ( unsigned long arg ), char
*stack_ptr, long stack_size, unsigned long parameter, int createFlags,
unsigned long *threadIDp );
```

Arguments

ap	this thread entry pointer
stack_ptr	this thread stack pointer
stack_size	this thread stack size
parameter	the argument passing to **ap()**
createFlags	currently not used, must be set to 0
threadIDp	pointer to a buffer to return the new thread ID

Description

create a thread

Return Value

0	OK
-1	no resource
-2	error argument

sys_ThreadResume	resume a thread	

Syntax

```
#include  <sdktask.h>
int  sys_ThreadResume ( unsigned long threadID );
```

Arguments

threadID	thread ID returned from **sys_ThreadCreate()**

Description

resume a thread

Return Value

0	OK
-1	error threadID

sys_ThreadState	get a thread state	

Syntax

```
#include  <sdktask.h>
int  sys_ThreadState ( unsigned long threadID );
```

Arguments

threadID	thread ID returned from **sys_ThreadCreate()**

Description

get a thread state

Return Value

1	suspend
0	running
-1	error threadID

sys_ThreadSuspend	suspend a thread	

Syntax

```
#include  <sdktask.h>
int  sys_ThreadSuspend ( unsigned long threadID );
```

Arguments

threadID	thread ID returned from s**ys_ThreadCreate()**

Description

suspend a thread

Return Value

0	OK
-1	error threadID

Time Server Library Reference

sys_GetLocalTime	get local time	

Syntax

```
#include  <sdksys.h>
int  sys_GetLocalTime ( struct tm_local *tm );
```

Arguments

tm	local time information

Description

get local time

Return Value

0	OK
-1	fail

sys_SetLocalTime	set local time	

Syntax

```
#include  <sdksys.h>
int  sys_SetLocalTime ( struct tm_local *tm );
```

Arguments

tm	time information to be set

Description

set local time

Return Value

0	OK
-1	fail

sysc_getTimeServer	get IP address of time server	

Syntax

```
#include  <sdkconf.h>
int  sysc_getTimeServer ( char *buffer, int bufsize );
```

Arguments

buffer	pointer to the buffer to retrieve the time server address
bufsize	the size of **buffer** (bytes)

Description

get IP address of time server used to synchronize the system time

Return Value

>= 0	number of bytes placed in buffer
-1	fail

sysc_getTimeZone	get the time offset used by time synchronization	

Syntax

```
#include  <sdkconf.h>
long  sysc_getTimeZone ( void );
```

Arguments

N/A

Description

This function will retrieve the time offset from local time zone to UTC. The offset is used in time synchronization. The returned value is the time offset to UTC (seconds). For example, a GMT +8:00 time zone has an offset of 28800 seconds.

Return Value

>= 0	time zone index number
-1	fail

sysc_getTZoneIndex	get the time zone of local system	

Syntax

```
#include  <sdkconf.h>
int  sysc_getTZoneIndex ( void );
```

Arguments

N/A

Description

retrieve the time zone of local system; all time zones are listed later in this chapter

Return Value

>= 0	time zone index number
-1	fail

sysc_setTimeServer	set IP address of time server that is used to synchronize the system time	

Syntax

```
#include  <sdkconf.h>
int  sysc_setTimeServer ( char *buffer, int bufsize );
```

Arguments

buffer	the new time server address
bufsize	the length of server address in buffer (bytes)

Description

The NE-4100-P can synchronize its system time with a remote NTP server. To enable this function, the NTP server address must be specified. A software timer is used to simulate a real time clock. NTP is used to synchronize the date and time of the internal clock with time server. If time information cannot be obtained due to network trouble, the system time will be set to Jan.1, 2000.

Return Value

>= 0	number of bytes copied from buffer to system configuration
-1	fail

sysc_setTimeZone	set the time offset used for time synchronization	

Syntax

```
#include  <sdkconf.h>
void  sysc_setTimeZone ( long tz );
```

Arguments

tz	the time offset from local time zone to UTC

Description

This function will set the time offset from local time zone to UTC. The offset will be used in time synchronization. The offset is measured in seconds. For example, set this value to 28800 for a GMT +8:00 time zone.

Return Value

N/A

sysc_setTZoneIndex	set the time zone of local system	

Syntax

```
#include  <sdkconf.h>
void  sysc_setTZoneIndex ( int index );
```

Arguments

index	time zone index number

Description

This function will set the time zone of local system. All time zones are listed later in this chapter. Note that time zone indicated by **index** is used for human readability and takes no effect when synchronizing the local time with the time server. You must call **sysc_setTimeZone()** to set the actual time offset.

Return Value

N/A

Time Zone Offsets Index

The hour offsets for different time zones are listed below. You will need this information when setting the time zone for automatic date/time synchronization. GMT stands for Greenwich Mean Time, which is the global time that all time zones are measured from.

Index	Offset	Status	Region
1	-43200	(GMT-12:00)	Eniwetok, Kwajalein
2	-39600	(GMT-11:00)	Midway Island, Samoa
3	-36000	(GMT-10:00)	Hawaii
4	-32400	(GMT-09:00)	Alaska
5	-28800	(GMT-08:00)	Pacific Time (US & Canada); Tijuana
6	-25200	(GMT-07:00)	Arizona
7	-25200	(GMT-07:00)	Mountain Time (US & Canada)
8	-21600	(GMT-06:00)	Central Time (US & Canada)
9	-21600	(GMT-06:00)	Mexico City, Tegucigalpa
10	-21600	(GMT-06:00)	Saskatchewan
11	-18000	(GMT-05:00)	Bogota, Lima, Quito
12	-18000	(GMT-05:00)	Eastern Time (US & Canada)
13	-18000	(GMT-05:00)	Indiana (East)
14	-14400	(GMT-04:00)	Atlantic Time (Canada)
15	-14400	(GMT-04:00)	Caracas, La Paz
16	-14400	(GMT-04:00)	Santiago
17	-12600	(GMT-03:30)	Newfoundland
18	-10800	(GMT-03:00)	Brasilia
19	-10800	(GMT-03:00)	Buenos Aires, Georgetown
20	-7200	(GMT-02:00)	Mid-Atlantic
21	-3600	(GMT-01:00)	Azores, Cape Verde Is.
22	0	(GMT)	Casablanca, Monrovia
23	0	(GMT)	Greenwich Mean Time: Dublin, Edinburgh, Lisbon, London
24	3600	(GMT+01:00)	Amsterdam, Berlin, Bern, Rome, Stockholm, Vienna
25	3600	(GMT+01:00)	Belgrade, Bratislava, Budapest, Ljubljana, Pragu
26	3600	(GMT+01:00)	Brussels, Copenhagen, Madrid, Paris, Vilnius
27	3600	(GMT+01:00)	Sarajevo, Skopje, Sofija, Warsaw, Zagreb
28	7200	(GMT+02:00)	Athens, Istanbul, Minsk
29	7200	(GMT+02:00)	Buchares
30	7200	(GMT+02:00)	Cairo
31	7200	(GMT+02:00)	Harare, Pretoria
32	7200	(GMT+02:00)	Helsinki, Riga, Tallinn
33	7200	(GMT+02:00)	Jerusalem
34	10800	(GMT+03:00)	Baghdad, Kuwait, Riyadh
35	10800	(GMT+03:00)	Moscow, St. Petersburg, Volgograd
36	10800	(GMT+03:00)	Mairobi
37	12600	(GMT+03:30)	Tehran
38	14400	(GMT+04:00)	Abu Dhabi, Muscat
39	14400	(GMT+04:00)	Baku, Tbilisi

Index	Offset	Status	Region
40	16200	(GMT+04:30)	Kabul
41	18000	(GMT+05:00)	Ekaterinburg
42	18000	(GMT+05:00)	Islamabad, Karachi, Tashkent
43	19800	(GMT+05:30)	Bombay, Calcutta, Madras, New Delhi
44	21600	(GMT+06:00)	Astana, Almaty, Dhaka
45	21600	(GMT+06:00)	Colombo
46	25200	(GMT+07:00)	Bangkok, Hanoi, Jakarta
47	28800	(GMT+08:00)	Beijing, Chongqing, Hong Kong, Urumqi
48	28800	(GMT+08:00)	Perth
49	28800	(GMT+08:00)	Singapore
50	28800	(GMT+08:00)	Taipei
51	32400	(GMT+09:00)	Osaka, Sapporo, Tokyo
52	32400	(GMT+09:00)	Seoul
53	32400	(GMT+09:00)	Yakutsk
54	34200	(GMT+09:30)	Adelaide
55	34200	(GMT+09:30)	Darwin
56	36000	(GMT+10:00)	Brisbane
57	36000	(GMT+10:00)	Canberra, Melbourne, Sydney
58	36000	(GMT+10:00)	Guam, Port Moresby
59	36000	(GMT+10:00)	Hobart
60	36000	(GMT+10:00)	Vladivostok
61	39600	(GMT+11:00)	Magadan, Solomon Is., New Caledonia
62	43200	(GMT+12:00)	Auckland, Wllington
63	43200	(GMT+12:00)	Fiji, Kamchatka, Marshall Is.

A
External Function Calls

We have tested the following standard Turbo C string functions with the SDK, and have verified that they can be used without any problem.

Function Name	Description
`strcat()`	append a string
`strchr()`	find a character in a string
`strcmp()`	compare strings
`strcpy()`	copy a string
`strlwr()`	convert a string to lowercase
`strupr()`	convert a string to uppercase
`strlen()`	get the length of a string
`atoi()`	convert strings to integer
`atol()`	convert strings to long
`itoa()`	convert an integer to a string
`ltoa()`	convert a long integer to a string

Note that to use these string functions, you must link to the **`cl.lib`** library file, with a **`tlink`** command similar to the one shown below:

```
%path:>tlink /t /s c0sdk+ap, ap, ap, moxa_sdk+c:\tc\lib\cl.lib
```

ATTENTION

You must use the complete path to link to the to the **`cl.lib`** library file.

If you would like to use other Turbo C standard functions, we cannot guarantee that they will work with the SDK. (When using Borland C, use the same method as for Turbo C.)

ATTENTION

There are several types of function calls that must not be used in programs for the NE-4100-P:

- system I/O functions such as **`printf()`**
- system interrupt function **`open()`**
- system memory allocate function **`malloc()`**