NPort

Programmable Communication Gateway
API Reference

Second Edition, April 2003

Moxa Technologies Co., Ltd.
Tel: +866-2-8919-1230
Fax: +886-2-8919-1231
www.moxa.com
support@moxa.com.tw

MOXA PCG API Reference

Copyright Notice

Trademarks

Disclaimer

MOXA Internet Services

Customer satisfaction is our number one concern. To ensure that customers receive the full benefit of our products, Moxa Internet Services has been set up to provide technical support, driver updates, product information, and user's manual updates.

The following services are provided:

E-mail for technical support

address: support@moxa.com.tw

World Wide Web (WWW) site for product information

address: http://www.moxa.com
or
http://www.moxa.com.tw

Table of Contents

1
Overview

The purpose of this **Moxa PCG API Reference** is to give MOXA PCG (Programmable Communication Gateway) programmers a complete reference guide to the various function calls that are available. You may also refer to the companion guide, *MOXA Programmable Communication Gateway Programmer's Guide*. **API** stands for **Application Programming Interface**, which includes necessary function calls and linking libraries.

Moxa API Quick Reference

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

Function Name	**Brief function introduction**	**Function Attributes**
Language Format		
Syntax	**#include <header file name>**	
	Function call	
Arguments	*Variable names* Brief description of variables	
Description	Detailed function call description.	
Return Value	**Return Code #1** Description of return code	
	Return Code #2 Description of return code	

To give you a specific example, we show here the function **sio_oqueue**, which is 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 in both the system's output buffer and the driver's output buffer.**	**Port Status**
C Format		
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 (in bytes) still remaining in the driver's output buffer.	
	SIO_BADPORT Port was not open in advance.	

2

SDK API Overview

SDK stands for **Software Development Kit**, and includes not only the SDK APIs, but also SDK Utilities, the Windows utility used by the programmer to communicate with PCG, plus several detailed example programs. You may also refer to the companion guide, *MOXA Programmable Communication Gateway Programmer's Guide* for additional information about using the utility.

In order to make the SDK library easier to use, the function calls are divided into six groups, based on their attributes. The six groups are **Serial I/O API**, **BSD Socket API**, **Simplified Socket API**, **System Control API**, **Flash ROM Access API**, and **Debug API**.

By keeping these 6 groups of APIs in mind, programmers can more easily program the PCG to meet the needs of their application, and set up the PCG to operate as needed.

In this chapter, we give a brief introduction to all function calls for each of the six groups so that programmers can get a good overview of all of the APIs. For detailed usage of each API, refer to the following six sections:

- Serial I/O API Overview
- BSD Socket API Overview
- Simplified Socket API Overview
- System Control API Overview
- Flash ROM Access API Overview
- Debug API Overview

Serial I/O API

In this section, we categorize the serial I/O library routines according to their function (**Port Control**, **Data Input**, **Data Output**, **Port Status Inquiry**, **Event Control**, and **Miscellaneous**). See Section 3-1 for a more detailed description of these functions.
You should also note you must include the header file **sdksio.h** in your source code that when calling these functions (see the example source code for details of how to include a header file).

Port Control This category includes functions to open serial ports, set communication parameters, and control signal lines.	
Function Name	**Description**
sio_open	Start receiving/transmitting data.
sio_close	Stop receiving/transmitting data.
sio_ioct1	Set port baud rate, parity, etc.
sio_flowctrl	Set port H/W and/or S/W flow control.
sio_flush	Flush input and/or output buffer.
sio_DTR	Set DTR state.
sio_RTS	Set RTS state.
sio_lctrl	Set both DTR and RTS states.
sio_baud	Set baud rate using the actual speed value.

Data Input This category includes functions to read data from the COM port.	
Function Name	**Description**
sio_getch	Read one character at a time from driver's input buffer.
sio_read	Read a block of data from the driver's input buffer.
sio_SetReadTimeouts	Set timeouts for sio_read().
sio_GetReadTimeouts	Get timeouts for sio_read().
sio_AbortRead	Abort when reading a block of data for sio_read().
sio_linput	Read a block of data ending with a termination character.

Data Output This category includes functions to write data to the serial port.	
Function Name	**Description**
sio_putch	Write one character at a time to driver's output buffer.
sio_write	Write a block of data (probably only a partial block will be written).
sio_SetWriteTimeouts	Set timeouts for sio_write().
sio_GetWriteTimeouts	Get timeouts for sio_write().
sio_AbortWrite	Abort when writing a block of data for sio_write().

Port Status Inquiry This category includes functions to query the communication status from the serial port.	
Function Name	**Description**
sio_lstatus	Get line status.
sio_iqueue	Size of data accumulated in driver's input buffer.
sio_oqueue	Size of data not yet sent out (still kept in driver's output buffer).
sio_Tx_hold	Check why data could not be transmitted.
sio_getbaud	Get the baud rate setting.
sio_getmode	Get the settings for parity, data bits, etc.
sio_getflow	Get the H/W and S/W flow control settings.
sio_data_status	Check if errors occur when receiving data.

Event Control This category includes functions to set the communication event service routines for the serial port.	
Function Name	**Description**
sio_term_irq	Set event service routine when termination character is received.
sio_cnt_irq	Set event service routine when a certain amount of data is received.
sio_modem_irq	Set event service routine when line status is changed.
sio_break_irq	Set event service routine when break signal is received.
sio_Tx_empty_irq	Set event service routine when transmit buffer is empty.

Miscellaneous This category includes special COM port functions.	
Function Name	**Description**
sio_break	Send out BREAK signal.
sio_break_ex	Send out BREAK signal.
sio_ActXon	Causes transmission to act as if an XON character has been received.
sio_ActXoff	Causes transmission to act as if an XOFF character has been received.

BSD Socket API

In this section, we categorize the BSD Socket library routines according to their function (**Socket Control**, **Data Input/Output**, **Socket Status Inquiry**, and **Miscellaneous**). See Section 3-2 for a more detailed description of these functions.

You should also note that the header file **sdksock.h** must be included in your source code when calling these functions (see the example source code for details of how to include a header file).

Socket Control This category includes functions to open TCP sockets, and set and retrieve communication parameters.	
Function Name	**Description**
accept	An incoming connection is acknowledged and associated with an immediately created socket. The original socket is returned to the 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.
ioctlsocket	Provides 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.
getsockopt	Retrieve options associated with the specified socket.

Data Input/Output This category includes functions to read/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.

Socket Status Inquiry This category includes functions to query the communication status from the socket.	
Function Name	**Description**
getpeername	Retrieve the name of the peer connected to the specified socket.
getsockname	Retrieve the current name for the specified socket.
gethostname	Retrieve the name of the local host.
gethostbyname	Retrieve the name(s) and address corresponding to a host name.

Miscellaneous This category includes special socket functions.	
Function Name	**Description**
htonl	Convert a 32-bit quantity from host byte order to network byte order.
htons	Convert a 16-bit quantity from host byte order to network byte order.
inet_addr	Convert a character string representing a number in the Internet standard "." notation to an Internet address value.
inet_ntoa	Convert an Internet address value to an ASCII string in "." notation (i.e., "a.b.c.d").
ntohl	Convert a 32-bit quantity from network byte order to host byte order.
ntohs	Convert a 16-bit quantity from network byte order to host byte order.

Simplified Socket API

In this section, we categorize the Simplified Socket library routines according to their function (**Socket Control**, **Data Input/Output**, and **Socket Status Inquiry**). See Section 3-3 for a more detailed description of these functions.

You should also note that calling these functions requires that the header file **sdknet.h** must be included in your source code (see the example source code for details of how to include a header file).

Socket Control This category includes functions to open TCP sockets, and set and retrieve communication parameters.	
Function Name	**Description**
tcp_open	Open a local TCP port.
tcp_close	Close a local TCP port.
tcp_connect	Connect to specific host IP and port.
tcp_listen	Place a socket in a state where it is listening for an incoming connection.
tcp_listento	Listen for a specific incoming connection.
tcp_connect_nowait	Connect to a specific host IP and port no wait.
tcp_listen_nowait	Place a socket in a state where it is listening for an incoming connection no wait.
tcp_listento_nowait	Listen for a specific incoming connection no wait.
udp_open	Open a local UDP port.
udp_close	Close a local UDP port.

Data Input/Output This category includes functions to read/data from the socket.	
Function Name	**Description**
tcp_send	Send data out through a connected socket.
tcp_recv	Receive data from a connected socket.
udp_send	Send data to a specific destination.
udp_recv	Receive data from a specific source address.

Socket Status Inquiry This category includes functions to query the communication status of the socket.	
Function Name	**Description**
tcp_ofree	Size of free space in the TCP driver's input buffer.
tcp_iqueue	Get the size of data accumulated in the TCP driver's input buffer.
tcp_get_remote	Get connected host IP and port.
tcp_state	Get TCP state.
udp_ofree	Size of free space in the UDP driver's input buffer.
udp_iqueue	Get the size of data accumulated in the UDP driver's input buffer.

Port Status Inquiry This category includes functions to open TCP sockets, and set and retrieve communication parameters.	
Function Name	**Description**
net_get_IP	Get local IP address.
net_get_netmask	Get local subnet mask.
net_get_gateway	Get local default gateway.
net_get_MAC_address	Get MAC address.

System Control API

This section presents the system information library routines, and gives a brief description of each routine. For a more detailed description of these routines, see Section 3-4.
You should also note that the header file **sdksys.h** must be included in your source code when calling these functions (see the example source code for details of how to include a header file).

Function Name	Description
sys_clock_s	Read the server's time (in seconds), measured from power-up.
sys_clock_ms	Read the server's time (in milliseconds) measured from power-up.
sys_sleep_ms	Task sleep time (milliseconds).
sys_timeout	Set the timeout event service routine.
sys_get_info	Get server's general information.
sys_enable_watchdog	Enable watchdog.
sys_disable_watchdog	Disable watchdog.
sys_get_WatchdogStatus	Get watchdog status.
sys_restart_system	Restart system.
sys_restart_UserAP	Restart user AP.
sys_set_RegisterID	Set AP ID.
sys_get_SerialType	Get current async port interface signal type.
sys_set_SerialType	Set the async port interface signal type.
sys_event_suspend	Suspend interrupt.
sys_event_resume	Resume interrupt.
sys_exit	Exit application.

Flash ROM Access API

This section presents the Flash Library routines, and gives a brief description of each routine. For a more detailed description of these routines, see Section 3-5. You should also note that the header file **sdkflash.h** must be included in your source code when calling these functions (see the example source code for details of how to include a header file).

Function Name	Description
flash_erase	Erase flash-ROM.
flash_length	Get current data length in the flash-ROM.
flash_write	Write data to the flash-ROM.
flash_read	Read data from the flash-ROM.

Debug API

This section presents the Debug Library routines, and gives a brief description of each routine. For a more detailed description of these routines, see Section 3-6. You should also note that the header file **sdkdbg.h** must be included in your source code when calling these functions (see the example source code for details of how to include a header file).

Function Name	Description
dbg_put_ch()	Print out a character for debugging.
dbg_put_block()	Print out a block of data for debugging.
dbg_put_word()	Print out a 2-byte unsigned integer value for debugging.
dbg_put_doubleword()	Print out a 4-byte unsigned long value for debugging.
dbg_put_word_hex()	Print out a 2-byte unsigned integer value with HEX format 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 with the a.b.c.d format for debugging.
dbg_put_string()	Print out a string for debugging.

3

SDK API Reference

3-1 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 was not open in advance.	

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 SIO_ABORT_WRITE.	
Return Value	SIO_OK OK SIO_BADPORT Port was not already open	

sio_ActXoff	**This function causes transmission to 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_Act_Xon	**This function causes transmission to 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 was not open in advance.	

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, 19200	
Description	Set baud rate using the actual speed value.	
Return Value	SIO_OK OK SIO_BADPORT Port was not open in advance.	

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 until the time has expired.	
Return Value	SIO_OK OK SIO_BADPORT Port was not open in advance. SIO_BADPARM Bad parameter	

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 in milliseconds	
Description	Sends out a break signal. This function will block transmission until the time has expired, and is the same as sio_break(), except that the time unit is measured in milliseconds.	
Return Value	SIO_OK OK SIO_BADPORT Port was not open in advance. SIO_BADPARM Bad parameter.	

sio_break_irq	Set an event service routine for the case 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 for the case 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 was not open in advance.	

sio_close	**Disable a serial port.**	**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/transmit data.	
Return Value	SIO_OK OK SIO_BADPORT Port was not open in advance.	

sio_cnt_irq	**Set an event service routine for the case 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 for the case 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 event service routine.	
Return Value	SIO_OK OK SIO_BADPORT Port was not open in advance.	

sio_data_status	**Check if an 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 occurred
> 0	bit 0 on - parity error bit 1 on - framing error bit 2 on - overrun error bit 3 on - overflow error
SIO_BADPORT	Port was not open in advance.

sio_DTR	**Set the 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 was not open in advance.
SIO_BADPARM	Bad parameter

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 was not open in advance.
SIO_BADPARM	Bad parameter

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 was not open in advance.
SIO_BADPARM	Bad parameter

sio_getbaud	**Get the 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. For example, a return value of 9600 means 9600 bps whereas 200 means 200 bps.	
Return Value	> 0	the actual baud rate
	SIO_BADPORT	Port was not open in advance.

sio_getch	**Read one character from the 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	The ASCII code of the character received.
	SIO_BADPORT	Port was not open in advance.
	SIO_NODATA	No data to read.
	SIO_BADPARM	Bad parameter.

sio_getflow	**Get the 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. See the sio_flowctrl() function.	
Return Value	>=0	bit 0 = l CTS flow contro
		bit 1 = RTS flow control
		bit 2 = Tx XON/XOFF flow control
		bit 3 = Rx XON/XOFF flow control
	SIO_BADPORT	Port was not open in advance.

sio_getmode	Get the 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	mode (see sio_ioctl())
SIO_BADPORT	Port was not open in advance.

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	Total timeout values
IntervalTimeouts	Interval timeout values

Description Get timeout values for sio_read and sio_getch.

Return Value

SIO_OK	OK
SIO_BADPORT	Port was not open in advance.

sio_GetWriteTimeouts	Get timeout values 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	Total timeout values

Description Get timeout values for sio_write() and sio_putch().

Return Value

SIO_OK	OK
SIO_BADPORT	Port was not already open

sio_ioctl	Control the settings of the 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 = bit_5
	0x01 = bit_6
	0x02 = bit_7
	0x03 = bit_8
stop_bit (bit 2) =	0x00 = stop_1
	0x04 = stop_2
parity (bits 3,4 5) =	0x00 = none
	0x08 = odd
	0x18 = even
	0x28 = mark
	0x38 = space

Description

Control 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 was not open in advance.
SIO_BADPARM	Bad parameter

sio_iqueue	**Get the size of data accumulated in the system's input buffer and driver's input buffer.**	**Port Status**

Syntax **#include <sdksio.h>**

long sio_iqueue (int *port*)

Arguments *port* Async serial port number

Description Get the size of data accumulated in the system's input buffer and driver's input buffer. (User must be aware of the fact that there may be a few characters still in the RS-232 UART chip and not yet known when sio_iqueue() returns a zero value.)

Return Value

>= 0	data in input buffer (bytes)
SIO_BADPORT	Port was not open in advance.

sio_lctrl	**Set both the 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 was not open in advance.
SIO_BADPARM	Bad parameter

sio_linput	Read a block of data from the driver's input buffer .	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
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
= 0	no data received
SIO_BADPORT	Port was not open in advance.
SIO_BADPARM	Bad parameter.

sio_lstatus	Get the status of the 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	line status Bit 0 – S_CTS Bit 1 – S_DSR Bit 2 – S_RI Bit 3 – S_CD
SIO_BADPORT	Port was not open in advance.

sio_modem_irq — **Set an event service routine for the case when the line status is 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 an event service routine for the case when the 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 was not open in advance.

sio_ofree — **Get the length of free space in the driver's output buffer.** — **Port Status**

Syntax

#include <sdksio.h>

long sio_ofree (int *port*)

Arguments

port Async serial port number

Description

Get the length of free space in the driver's output buffer.

Return Value

>= 0	free space in output buffer (bytes)
SIO_BADPORT	Port was not open in advance.

sio_open — **Enable a serial port for data transmitting/receiving.** — **Port Control**

Syntax

#include <sdksio.h>

int sio_open (int *port*)

Arguments

port Async serial port number for NPort-4511 is always 1.

Description

Enable a serial port for data transmitting/receiving. After calling sio_open, the initial status of this COM port is the same as the last setting or configuration setting.

Return Value

>= 0	Open action was successful, and this return value is a descriptor referencing the port. The 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 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 (in bytes) still remaining in the driver's output buffer.
SIO_BADPORT	Port was not open in advance.

sio_putch	**Write a character into the driver's output buffer.**	**Data Output**

Syntax **#include <sdksio.h>**

int sio_putch (int *port*, int *term*)

Arguments

port Async serial port number

term character (0 - 255)

Description

Write a character into the driver's output buffer.

Return Value

SIO_OK	OK
SIO_BADPORT	Port was not already open
SIO_BADPARM	Bad parameter
SIO_ABORT_WRITE	User abort blocked write
SIO_WRITETIMEOUT	Write timeout has occurred

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

Description

Read data from the driver's input buffer. If the length of data in the driver's input buffer is less than the user's buffer, then all data in 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_set_ReadTimeout() 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
= 0	no data received
SIO_BADPORT	Port was not open in advance.
SIO_BADPARM	Bad parameter.

sio_RTS	Set the RTS state of a 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 the RTS state of a port.

Return Value

SIO_OK	OK
SIO_BADPORT	Port was not open in advance.
SIO_BADPARM	Bad parameter.
SIO_RTS_BY_HW	Can't control the port because it is set as auto H/W flow control by sio_flowctrl().

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
IntervalTimeouts	Interval timeout values

Description Set timeout values for sio_read and sio_getch. The default *TotalTimeouts* value is MAXDWORD and the *IntervalTimeouts* value is 0, which enables sio_read to return immediately.

Return Value

SIO_OK	OK
SIO_BADPORT	Port was not open in advance.

sio_SetWriteTimeouts	Set timeout values 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 values

Description Set timeout values for sio_write() and sio_putch(). The default value is 0, which enables sio_write() to always block until it is finished writing data.

The value 0xFFFFFFFF 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 was not already open
SIO_BADPARM	Bad parameter

sio_term_irq	**Set an event service routine for the case 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 code

Description

Set an event service routine for the case 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 was not open in advance.

sio_Tx_empty_irq	**Set an event service routine for the case when the last character in the output buffer was sent.**	**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 for the case when the last character in the output buffer was sent. When the Tx empty signal is encountered, the system will call the event service routine.

Return Value

SIO_OK	OK
SIO_BADPORT	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; could not transmit data because CTS is low bit 1 on; could not transmit data because XOFF char received
SIO_BADPORT	Port was not open in advance.

sio_write	**Write a block of data to the 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

Description

Write 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.
sio_set_WriteTimeout() can be used to set timeouts for sio_write(). SIO_WRITETIMEOUT will be returned for sio_write() when write timeouts.
sio_abort_write() can be used to abort any blocked sio_write() with return value SIO_ABORT_WRITE.

Return Value

>= 0	length of data transmitted
SIO_BADPORT	Port was not already open
SIO_BADPARM	Bad parameter
SIO_ABORT_WRITE	User abort blocked write
SIO_WRITETIMEOUT	Write timeout has occurred

accept	**Accept a connection on a socket.**
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. The 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 (in bytes) of the address returned. This call is used with connection-based socket types such as SOCK_STREAM. If *addr* and/or *addrlen* are equal to NULL, then no information about the remote address of the accepted socket is returned.

Return Value

If no error occurs, accept() returns a value of type int which is a descriptor for the accepted packet. Otherwise, a value of -1 is returned, and the global variable *errno* contains one of the following values.
The integer referred to by *addrlen* initially contains the amount of space pointed to by *addr*. On return it will contain the actual length in bytes of the address returned.

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 argument 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.

See Also

bind(), connect(), listen(), select(), socket()

Warnings

bind — Associate a local address with a socket.

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	The length of the name.

Description

This routine is used on an unconnected datagram or stream socket, before subsequent connect()'s or listen()'s. 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 (set implicitly to UDP or TCP, respectively), 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 Windows Sockets implementation will assign a unique port to the application with a value between 1024 and 30000. The application may use getsockname() after bind() to learn 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 no error occurs, bind() returns 0. Otherwise, it returns –1, and the global variable *errno* contains 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.

See Also

connect(), listen(), getsockname(), setsockopt(), socket().

closesocket — **Close a socket.**

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 Windows Sockets 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 no error occurs, closesocket() returns 0. Otherwise, it returns -1, and the global variable *errno* contains one of the following values.

Error Codes EBADF The descriptor is not a socket.

See Also accept(), socket(), ioctlsocket(), setsockopt().

connect — Establish a connection to a peer.

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	The length of the *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/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

If no error occurs, connect() returns 0. Otherwise, it returns -1, and the global variable *errno* contains one of the following values.

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

See Also

accept(), bind(), getsockname(), socket(), select().

gethostbyname — **Get host information corresponding to a hostname.**

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 which 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, unused currently.
h_addrtype	The type of address being returned; this is always AF_INET.
h_length	The length, in bytes, this is always 4
h_addr_list	A NULL-terminated list of addresses for the host. Addresses are returned in network byte order.

The pointer returned points to a structure that is allocated by NPort Server. The application must not modify this structure or free any of its components.

Return Value

If no error occurs, gethostbyname() returns a pointer to the hostent structure described above. Otherwise it returns a NULL pointer.

See Also

gethostname()

gethostname	**Return the standard host name for the local machine.**
Syntax	**#include <sdksock.h>** **int gethostname (char *_name_, int _namelen_);**
Arguments	*name* A pointer to a buffer that will receive the host name. *namelen* The 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 sockets implementation—it is a simple host name. However, it is guaranteed that the name returned will be successfully parsed by gethostbyname().
Return Value	If no error occurs, gethostname() returns 0. Otherwise, it returns –1, and the global variable *errno* contains one of the following values.
Error Codes	WSAEFAUL The *namelen* parameter is too small or the name argument pointer is invalid.
See Also	gethostbyname()

getpeername	**Get the address of the peer to which a socket is connected.**
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 size 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 in bytes.

Return Value	If no error occurs, getpeername() returns 0. Otherwise, it returns –1, and the global variable *errno* contains one of the following values.	
Error Codes	EFAULT	The *name* argument pointer is invalid or *namelen* argument is not large enough.
	ENOTCONN	The socket is not connected.
	EBADF	The descriptor is not a socket.
See Also	bind(), socket(), getsockname().	

getsockname **Get the local name for a socket.**

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 size 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 in 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 no error occurs, getsockname() returns 0. Otherwise, it returns -1, and the global variable *errno* contains one of the following values.	
Error Codes	EFAULT	The address of *name* or *namelen* argument is not large enough.
	EBADF	The descriptor is not a socket.
See Also	bind(), socket(), getpeername()	

getsockopt **Retrieve a socket option.**

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. Options affect socket operations, such as whether an operation blocks or not, the routing of packets, out-of-band data transfer, etc.

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

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 no error occurs, getsockopt() returns 0. Otherwise, it returns –1, and the global variable *errno* contains one of the following values.

Error Codes

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

See Also

setsockopt(), socket()

htonl	**Convert an unsigned long from host to network byte order.**
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.
See Also	htons(), ntohl(), ntohs().

htons	**Convert an unsigned short from host to network byte order.**
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.
See Also	htonl(), ntohl(), ntohs().

inet_addr	**Convert a string containing a dotted address into an in_addr.**
Syntax	**#include <sdksock.h>** **unsigned long inet_addr (char *_cp_);**
Arguments	*cp* A character string representing a number expressed in the Internet 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 Values specified using the "." notation take the following forms: a.b.c.d 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. Note: The following notations are only used by Berkeley, and nowhere else on the Internet. In the interests of compatibility with their software, they are supported as specified.

Return Value	If no error occurs, 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.
See Also	inet_ntoa()

inet_ntoa **Convert a network address into a string in dotted format.**

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 sockets API call within the same thread, but no longer.
Return Value	If no error occurs, inet_ntoa() returns a char pointer to a static buffer containing the text address in standard "." notation. Otherwise, it returns NULL. The data should be copied before another Windows Sockets call is made.
See Also	inet_addr().

ioctlsocket	**Control the mode of a socket.**
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 communications subsystem. The following commands are supported: Command — Semantics FIONBIO — Enable or disable non-blocking mode on the socket *s*. *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). This is consistent with BSD sockets.
Compatibility	This function is a subset of ioctl() as used in Berkeley sockets.
Return Value	Upon successful completion, the ioctlsocket() returns 0. Otherwise, it returns –1, and the global variable *errn*o contains one of the following values.
Error Codes	EINVAL *cmd* is not a valid command, or *argp* is not an acceptable parameter for *cmd*, or the command is not applicable to the type of socket supplied. EBADF The descriptor *s* is not a socket.
See Also	socket(), setsockopt(), getsockopt().

listen **- Establish a socket to listen for incoming connection.**

Syntax

#**include <sdksock.h>**

int listen (int *s*, int *backlog*);

Arguments

s	A descriptor identifying a bound, unconnected socket.
backlog	The maximum length to which the queue of pending connections may grow.

Description

To accept connections, a socket is first created with socket(), a backlog for incoming connections is specified with listen(), and then the 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.

listen() attempts to continue to function rationally when there are no available descriptors. It will accept connections until the queue is emptied. If descriptors become available, a later call to listen() or accept() will refill the queue to the current or most recent "backlog" if possible, and then resume listening for incoming connections.

Compatibility

backlog is currently limited (silently) to 5. As in 4.3BSD, illegal values (less than 1 or greater than 5) are replaced by the nearest legal value.

Return Value

If no error occurs, listen() returns 0. Otherwise, it returns –1, and the global variable *errno* contains one of the following values.

Error Codes

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

See Also

accept(), connect(), socket().

ntohl	**Convert an unsigned long from network to host byte order.**
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.
See Also	htonl(), htons(), ntohs().

ntohs	**Convert an unsigned short from network to host byte order.**
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.
See Also	htonl(), htons(), ntohl().

recv	**Receive data from a socket.**
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 length of *buf*. *flags* Specifies the way in which the call is made.
Description	This function is used on connected datagram or stream sockets specified by the *s* parameter and is used to read incoming data. For sockets of type SOCK_STREAM, as much information as is 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 no error occurs, 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* contains one of the following values.

Error Codes		
	EBADF	The descriptor is not a socket.
	EFAULT	The buf argument pointer is invalid.
	EOPNOTSUPP	MSG_OOB was specified, but the socket is not of type SOCK_STREAM.
	ESHUTDOWN	The 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	The 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	The length argument is zero.

See Also recvfrom(), read(), ,recv(), send(), select(), socket()

recvfrom	**Receive a datagram and store the source address.**

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, as much information as is 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 no error occurs, 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* contains one of the following values.

Error Codes

EBADF	The descriptor is not a socket.
EFAULT	The *buf* argument pointer is invalid.
EOPNOTSUPP	MSG_OOB was specified, but the socket is not of type SOCK_STREAM.
ESHUTDOWN	The 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	The 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	The length argument is zero.

See Also

recv(), send(), socket().

select	**Determine the status of one or more sockets, waiting if necessary.**

Syntax

#include <sdksock.h>

int select (int *nfds*, fd_set **readfds*, fd_set **writefds*, fd_set **exceptfds*, struct timeval **timeout*);

Arguments

nfds	This argument is ignored and included only for the sake of compatibility.
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.

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 32). Internally, an fd_set is represented as an array of SOCKET's. The macros are:

FD_CLR(*s*, **set*)	Removes the descriptor *s* from *set*.
FD_ISSET(*s*, **set*)	Nonzero if *s* is a member of the *set*, 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.

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* contains one of the following values.

Error Codes

EINVAL	The refds, wrfds and exfds are all NULL.
EBADF	One of the descriptor sets contains an entry which is not a socket.

See Also

accept(), connect(), recv(), recvfrom(), send()

send	**Send data on a connected socket.**
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 no error occurs, 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* contains one of the following values.

Error Codes	EFAULT	The buf argument is not in a valid part of the user address space.
	ENOTCONN	The socket is not connected.
	EBADF	The descriptor is not a socket.
	EOPNOTSUPP	MSG_OOB was specified, but the socket is not of type SOCK_STREAM.
	ESHUTDOWN	The 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	The socket is marked as non-blocking and the requested operation would block.
	EFBIG	Data written exceeds system capacity.

See Also recv(), recvfrom(), socket(), sendto().

sendto	**Send data to a specific destination.**

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	The 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 no error occurs, 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* contains one of the following values.

Error Codes

EFAULT	The *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	The system had insufficient resources to perform the operation.
ENOTCONN	The socket is not connected (SOCK_STREAM only).
EBADF	The descriptor is not a socket.
EOPNOTSUPP	MSG_OOB was specified, but the socket is not of type SOCK_STREAM.
ESHUTDOWN	The 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	The socket is marked as non-blocking and the requested operation would block.
EINVAL	The socket has not been bound with baind().

See Also

recv(), recvfrom(), socket(), send().

setsockopt	**Set a socket option.**	
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 expedited data is received in the normal data stream, whether broadcast messages may be sent on the socket, 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 the an integer or structure that contains the desired value for the option, and *optlen* is the length of the integer or 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 (in seconds), and call setsockopt(). The timeout value should be in the interval between 0 to10 (in seconds). To enable SO_DONTLINGER (i.e., disable SO_LINGER) l_onoff should be set to zero and setsockopt() should be called.

By default, a socket may not be bound (see bind()) to a local address which is already in use. On occasion, however, it may be desirable to "re-use" an address in this way. Since every connection is uniquely identified by the combination of local and remote addresses, there is no problem with having two sockets bound to the same local address as long as the remote addresses are different. To inform the Windows Sockets implementation that a bind() on a socket should not be disallowed because the desired address is already in use by another socket, the application should set the SO_REUSEADDR socket option for the socket before issuing the bind(). Note that the option is interpreted only at the time of the bind(); it is therefore unnecessary (but harmless) to set the option on a socket which is not to be bound to an existing address, and setting or resetting the option after the bind() has no effect on this or any other socket.

An application may request that the Sockets implementation enable the use of "keep-alive" packets on TCP connections by turning on the SO_KEEPALIVE socket option. A Sockets implementation need not support the use of keep-alives; if it does, the precise semantics are implementation-specific, but should conform to section 4.2.3.6 of RFC 1122.

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

Value	Type	Meaning
SO_DONTLINGER	BOOL	Don't 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 no error occurs, setsockopt() returns 0. Otherwise, it returns -1, and the global variable *errno* contains 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.

See Also

bind(), getsockopt(), ioctlsocket(), socket().

shutdown	**Disable sends and/or receives on a socket.**
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, a 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.
Comments	shutdown() does not block regardless of the SO_LINGER setting on the socket. An application should not rely on being able to re-use a socket after it has been shut down. In particular, a Sockets implementation is not required to support the use of connect() on such a socket.
Return Value	If no error occurs, shutdown() returns 0. Otherwise, it returns –1, and the global variable *errno* contains one of the following values.
Error Codes	EINVAL *how* is not valid ENOTCONN The socket is not connected (SOCK_STREAM only). EBADF The descriptor is not a socket
See Also	connect(), socket().

socket	**Create a socket.**	
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:

Type Explanation

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. Out-of-band data may also be transmitted as described in send() and received as described in recv().

The communications protocols used to implement a 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 such a socket is connect()'ed to a specific peer, datagrams may be sent to that peer using send() and may be received from (only) this peer using recv().

Return Value

If no error occurs, socket() returns a descriptor referencing the new socket. Otherwise, it returns -1, and the global variable *errno* contains one of the following values.

Error Codes

EMFILE	No more file descriptors are available.
EPROTONOSUPPORT	The specified address family or protocol is not supported.

See Also

accept(), bind(), connect(), getsockname(), getsockopt(), setsockopt(), listen(), recv(), recvfrom(), select(), send(), sendto(), shutdown(), ioctlsocket().

net_get_gateway	Get local default gateway.
Syntax	**#include <sdknet.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.
Syntax	**#include <sdknet.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.
Syntax	**#include <sdknet.h>** **void net_get_MAC_address (u_char *mac);**
Arguments	*mac* Get MAC address data buffer pointer.
Description	Get MAC address.
Return Value	System copies the host MAC address to the *mac* input buffer.

net_get_netmask	Get local subnet mask.
Syntax	**#include <sdknet.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.**
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 O.K -1 error handle number.
See Also	

tcp_connect	**Connect to specific host IP and port.**
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 no. *tout* wait for TCP connection time out value: ms. 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 error argument. -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.**
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 No.
Description	Connect to specific host's IP and port no wait.
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.
See Also	

tcp_get_remote	**Get connected host's IP and port.**
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.
See Also	

tcp_iqueue	**Get the size of data accumulated in TCP driver's input buffer.**
Syntax	**#include <sdknet.h>** **int tcp_iqueue (int *handle*)**
Arguments	*handle* the value returned from tcp_open()
Description	Get the size 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	**Places a socket in a state where it is listening for an incoming connection.**
Syntax	**#include <sdknet.h>** **int tcp_listen (int *handle*, long *tout*);**
Arguments	*handle* the value return from tcp_open(). *tout* wait for listen time out value, unit is ms. 0 for 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 was opened by tcp_open(). -3 timeout counter reached. -4 error state; already connected.

tcp_listen_nowait	**Places a socket in a state where it is listening for an incoming connection no wait.**
Syntax	**#include <sdknet.h>** **int tcp_listen_nowait (int *handle*);**
Arguments	*handle* the value returned from tcp_open()
Description	Places a socket a state where it is listening for an incoming connection no wait.
Return Value	0 start to listen. -1 error handle number -2 this handle was opened by tcp_open(). -3 error state; already connected.

tcp_listento	**Listen for a specific incoming connection.**
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 don't case remote IP address. *rport* remote host TCP port No. 0 indicates don't case the TCP port number. *tout* wait for listen timeout value; unit is ms.
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 was opened by tcp_open(). -3 timeout counter reached. -4 error state; already connected.

tcp_listento_nowait **Listen for a specific incoming connection no wait.**

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 don't case remote IP address. *rport* remote host's TCP port No. 0 indicates don't case the 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 was opened by tcp_open(). -3 error state; already connected.

tcp_ofree **Size of free space in TCP driver's input buffer.**

Syntax	**#include <sdknet.h>** **int tcp_ofree (int *handle*)**
Arguments	*handle* the value returned from tcp_open()
Description	Size 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.**

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	**Receives data from a connected socket.**
Syntax	**#include <sdknet.h>** **int tcp_recv (int *handle*, char **buffer*, int *len*);**
Arguments	*handle* the value returned from tcp_open() *buffer* the receiveed data buffer pointer. *len* buffer length
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 on a connected socket.**
Syntax	**#include <sdknet.h>** **int tcp_send (int *handle*, char **buffer*, int *len*);**
Arguments	*handle* the value return from tcp_open() *buffer* the send out data buffer pointer. *len* data length
Description	Sends data on a connected socket.
Return Value	>=0 send out data length. -1 error handle number. -2 error argument. -3 TCP not connected.

tcp_state	**Get TCP state.**
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.**
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 the size of data accumulated in UDP driver's input buffer.**
Syntax	**#include <sdknet.h>** **int udp_iqueue (int *handle*)**
Arguments	*handle* the value returned from udp_open()
Description	Get the size 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	Size of free space in UDP driver's input buffer.
Syntax	**#include <sdknet.h>** **int udp_ofree (int *handle*)**
Arguments	*handle* the value returned from udp_open()
Description	Size 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.
Syntax	**#include <sdknet.h>** **int udp_open (int *port*)**
Arguments	*port* the local UDP port number
Description	Open a local UDP port.
Return Value	>=0 open handle. -1 open fail.

udp_recv	Receives data from a specific source address.
Syntax	**#include <sdknet.h>** **int udp_recv** **(int *handle*, u_long **rip*, int **rport*, char **buf*, int *len*);**
Arguments	*handle* the value return from udp_open(). *rip* recv from host IP address pointer. *rport* recv from host UDP port number pointer. *buf* recv data buffer pointer. *len* recv data buffer length.
Description	Receives data from a specific source address.
Return Value	>= 0 recv data length. -1 recv failed.

udp_send	**Sends data to a specific destination.**	
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	send to host IP address.
	rport	send to host UDP port number.
	buf	send data buffer pointer.
	len	send data length
Description	Sends data to a specific destination.	
Return Value	>= 0	sent out data length.
	-1	send failed.

sys_clock_ms	**Read the server's time (milliseconds) count from power-up.**
Syntax	**#include <sdksys.h>** **unsigned long sys_clock_ms (void);**
Arguments	N/A
Description	Read the server's time (milliseconds) count from power-up.
Return Value	This function returns server's time counter in milliseconds.
See Also	sys_clock_s()

sys_clock_s	**Read the server's time (seconds) count from power-up.**
Syntax	**#include <sdksys.h>** **unsigned long sys_clock_s (void);**
Arguments	N/A
Description	Read the server's time (seconds) count from power-up.
Return Value	This function returns server's time counter in seconds.
See Also	sys_clock_ms()

sys_disable_watchdog	**Disable watchdog.**
Syntax	**#include <sdksys.h>** **void sys_watchdog_disable(void);**
Arguments	N/A
Description	Disable watchdog.
Return Value	N/A

sys_enable_watchdog	**Enable watchdog.**
Syntax	**int sys_enable_watchdog (int *mode*);**
Arguments	*mode* 1 watch-dog reset timeout value is 335ms. 2 watch-dog reset timeout value is 419ms. 3 watch-dog reset timeout value is 671ms. 4 watch-dog reset timeout value is 838ms. 5 watch-dog reset timeout value is 1.34s. 6 watch-dog reset timeout value is 1.68s. 7 watch-dog reset timeout value is 2.68s. 8 watch-dog reset timeout value is 3.35s.
Description	Enable watchdog.
Return Value	0 OK -1 error argument.

sys_exit	**Exit application.**
Syntax	**#include <sdksys.h>** **Void sys_exit (void);**
Arguments	*N/A*
Description	To exit user application and return to kernel. It will let the user application stop it.
Return Value	N/A
See Also	N/A

sys_get_info — **Get server general information.**

Syntax

#include <sdksys.h>

int sys_get_info (struct sdk_sysinfo *_info_);

Arguments

info A pointer to a buffer that will receive the server general information.

Description

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

e.g., Ver 1.20.3, the main version is 1, sub_version is 20 and ext_version is 3.

```
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 */
};
```

Return Value	sys_get_info() returns the buffer data length of the information structure. Return of 0 indicates the argument is invalid.

sys_get_SerialType — **Get async port interface signal type.**

Syntax	**#include <sdksys.h>** **sys_get_SerialType (int port);**
Arguments	*port* Async serial port number
Description	Get async port interface signal type.
Return Value	0 RS-232 1 RS-422 2 RS-485 (2w) 3 RS-485 (4w) -1 Bad port

sys_get_WatchdogStatus — **Get watchdog status.**

Syntax	**#include <sdksys.h>** **int sys_get_WatchdogStatus (void);**
Arguments	N/A
Description	Get watchdog status.
Return Value	0 watch-dog timer is disabled. 1 watch-dog reset timeout value is 335ms. (refer to sys_watchdog_enable()) 8 watch-dog reset timeout value is 3.35s.

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.

Description
It let user the application ID.
User can get by DSCI. And 0x80000000 to 0xFFFFFFFF is reversed for MOXA only.
User just can use 0x00000000 to 0x7FFFFFFF. Your application starts to run to need first to call it.

Return Value N/A

See Also N/A

sys_set_SerialType — **Set async port interface signal type.**

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

Arguments
port Async serial port number
0 RS-232
1 RS-422
2 RS-485 (2w)
3 RS-485 (4w) *Note: Not supported by NPort 4511.*

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	**Task sleep time (milliseconds).**
Syntax	**#include <sdksys.h>** **int sys_sleep_ms (long *time_ms*);**
Arguments	*time_ms* Task sleep time in milliseconds.
Description	Task sleep time (milliseconds).
Return Value	This function always returns 0.
See Also	sys_clock_s(), sys_clock_ms()

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 in 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.
See Also	sys_clock_s(), sys_clock_ms(), sys_sleep_ms()..

sys_event_suspend	Suspend interrupt
Syntax	**#include <sdksys.h>** **void sys_event_suspend(int *type*, u_long *args*);**
Arguments	*type* interrupt type 0 → sys_timeout 1 → sio_term_irq 2 → sio_cnt_irq 3 → sio_modem_irq 4 → sio_break_irq 5 → sio_tx_Empty_irq *args* If "type" is 0, "args" is the timeouts interrupt subroutine. If "type" is not 0, "args" is the serial port number. Eq. 1 for NPort 4511's serial port.
Description	This function suspends the specific interrupt when the subroutine is called not, from specific interrupt, preventing unpredictable situations from occurring. This function is usually called at the beginning of a subroutine. For example, suppose brk_isr() is sio_break_irq() interrupt subroutine (ISR). If it is called from the main program instead of a break interrupt, this function will prevent unpredictable results.
Return Value	0 OK -1 Fail
See Also	sys_event_resume

sys_event_resume	**Resume interrupt**
Syntax	**#include <sdksys.h>** **void sys_event_resume(int *type*, u_long *args*);**
Arguments	*type* interrupt type 0 → sys_timeout 1 → sio_term_irq 2 → sio_cnt_irq 3 → sio_modem_irq 4 → sio_break_irq 5 → sio_tx_Empty_irq *args* If "type" is 0, "args" is the timeouts interrupt subroutine. If "type" is not 0, "args" is the serial port number. Eq. 1 for NPort 4511's serial port.
Description	This will resume the interrupt that is suspended by sys_event_suspend in subroutine. This function is usually called at the end of the subroutine.
Return Value	0 OK -1 Fail
See Also	sys_event_suspend

3-5 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 at the flash-ROM.**
Syntax	**#include <sdkflash.h>** **long flash_length (void)**
Arguments	N/A
Description	Get current data length at the flash-ROM
Return Value	>=0 Current data length at the flash-ROM. 0 after calling sys_flash_erase(). Max. length is 65536. (64 KB).

flash_read	**Read data from the flash-ROM.**
Syntax	**#include <sdkflash.h>** **Long flash_read** **(long offset, char *_buffer_, long _size_);**
Arguments	*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 the 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 65536
Description	Write data to the flash-ROM.
Return Value	>0 write length. -1 write failed -2 Need to erase the flash-ROM first.

3-6 Debug Library Reference

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 print out messages.
See Also	dbg_put_ch(), dbg_put_word(), dbg_put_doubleword(), dbg_put_word_hex(), dbg_put_doubleword_hex(), dbg_put_IP(), dbg_put_string.

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 print out messages
See Also	dbg_put_ch(), dbg_put_block(), dbg_put_word(), dbg_put_word_hex(), dbg_put_doubleword_hex(), dbg_put_IP(), dbg_put_string().

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 print out messages.
See Also	dbg_put_ch(), dbg_put_block(), dbg_put_word(), dbg_put_doubleword(), dbg_put_word_hex(), dbg_put_IP(), dbg_put_string().

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 the printed out messages.
See Also	dbg_put_block(), dbg_put_word(), dbg_put_doubleword(), dbg_put_word_hex(), dbg_put_doubleword_hex(), dbg_put_IP(), dbj_put_string().

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 print out messages.
See Also	dbg_put_ch(), dbg_put_block(), dbg_put_word(), dbg_put_doubleword(), dbg_put_word_hex(), dbg_put_doubleword_hex(), dbg_put_string().

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 print out messages.
See Also	dbg_put_ch(), dbg_put_block(), dbg_put_word(), dbg_put_doubleword(), dbg_put_word_hex(), dbg_put_doubleword_hex(), dbg_put_IP().

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 print out messages.
See Also	dbg_put_ch(), dbg_put_block(), dbg_put_doubleword(), dbg_put_word_hex(), dbg_put_doubleword_hex(), dbg_put_IP(), dbg_put_string().

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 print out messages.
See Also	dbg_put_ch(), dbg_put_block(), dbg_put_word(), dbg_put_doubleword(), dbg_put_doubleword_hex(), dbg_put_IP(), dbg_put_string().

4

External Function Calls for SDK

We have tested the following standard Turbo C string functions with 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 **cs.lib** library file, with a tlink command similar to the one shown here.

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

NOTE *It is important to keep in mind that you must use the **complete path** to link to the to the cs.lib library file.*

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

NOTE *There are several types of function calls that **must not** be used in programs for NPort-4511. They are:*

System I/O: such as printf()
System Interrupt: open()
System Memory Allocate: malloc()
