

# Network Enabler SDK 2 API Reference

---

Sixth Edition, September 2006

[www.moxa.com/product](http://www.moxa.com/product)



MOXA Technologies Co., Ltd.

Tel: +886-2-8919-1230

Fax: +886-2-8919-1231

Web: [www.moxa.com](http://www.moxa.com)

**MOXA Technical Support**

Worldwide: [support@moxa.com](mailto:support@moxa.com)

The Americas: [support@usa.moxa.com](mailto:support@usa.moxa.com)

# Network Enabler SDK 2 API Reference

The software described in this manual is furnished under a license agreement and may be used only in accordance with the terms of that agreement.

## Copyright Notice

Copyright © 2006 MOXA Technologies Co., Ltd.  
All rights reserved.  
Reproduction without permission is prohibited.

## Trademarks

MOXA is a registered trademark of The MOXA Group.  
All other trademarks or registered marks in this manual belong to their respective manufacturers.

## Disclaimer

Information in this document is subject to change without notice and does not represent a commitment on the part of MOXA.

MOXA provides this document “as is,” without warranty of any kind, either expressed or implied, including, but not limited to, its particular purpose. MOXA reserves the right to make improvements and/or changes to this manual, or to the products and/or the programs described in this manual, at any time.

Information provided in this manual is intended to be accurate and reliable. However, MOXA assumes no responsibility for its use or for any infringements on the rights of third parties that may result from its use.

This product may include unintentional technical or typographical errors. Changes are periodically made to the information herein to correct such errors, and these changes are incorporated into new editions of the publication.

# Table of Contents

<b>Chapter 1.</b>	<b>Overview .....</b>	<b>1-1</b>
	Sample Layout.....	1-2
<b>Chapter 2.</b>	<b>API List .....</b>	<b>2-1</b>
	Overview .....	2-2
	Serial I/O API .....	2-2
	Port Control.....	2-2
	Data Input.....	2-2
	Data Output.....	2-2
	Port Status .....	2-3
	Event Control.....	2-3
	Miscellaneous.....	2-3
	BSD Socket API .....	2-4
	Socket Control .....	2-4
	Data Input/Output.....	2-4
	Inquiry .....	2-4
	Miscellaneous.....	2-5
	Simplified Socket API.....	2-5
	Socket Control .....	2-5
	Data Input/Output.....	2-5
	Socket Inquiry.....	2-5
	Port Inquiry.....	2-6
	System Control API.....	2-6
	Flash ROM Access API.....	2-7
	Debug API.....	2-7
	DIO API.....	2-8
	Thread Control API .....	2-8
	Time Server API .....	2-8
<b>Chapter 3.</b>	<b>API Reference .....</b>	<b>3-1</b>
	Serial I/O Library Reference .....	3-2
	sio_AbortRead.....	3-2
	sio_AbortWrite .....	3-2
	sio_ActXoff .....	3-2
	sio_ActXon.....	3-3
	sio_baud.....	3-3
	sio_break.....	3-3
	sio_break_ex.....	3-4
	sio_break_irq.....	3-4
	sio_close .....	3-4
	sio_cnt_irq .....	3-5
	sio_data_status.....	3-5
	sio_DTR .....	3-6
	sio_flowctrl.....	3-6
	sio_flush .....	3-7
	sio_getbaud.....	3-7
	sio_getch.....	3-7
	sio_getflow .....	3-8
	sio_getmode.....	3-8
	sio_GetReadTimeouts.....	3-8

sio_GetWriteTimeouts.....	3-9
sio_ioctl.....	3-10
sio_iqueue.....	3-11
sio_lctrl.....	3-11
sio_linput.....	3-12
sio_lstatus.....	3-12
sio_modem_irq.....	3-13
sio_ofree.....	3-13
sio_open.....	3-13
sio_oqueue.....	3-14
sio_putch.....	3-14
sio_read.....	3-15
sio_RTS.....	3-15
sio_SetReadTimeouts.....	3-16
sio_SetWriteTimeouts.....	3-16
sio_term_irq.....	3-17
sio_Tx_empty_irq.....	3-17
sio_Tx_hold.....	3-18
sio_write.....	3-18
BSD Socket Library Reference.....	3-19
accept.....	3-19
bind.....	3-20
closesocket.....	3-21
connect.....	3-22
gethostbyname.....	3-23
gethostname.....	3-24
getpeername.....	3-24
getsockname.....	3-25
getsockopt.....	3-26
htonl.....	3-27
htons.....	3-27
inet_addr.....	3-28
inet_ntoa.....	3-28
ioctlsocket.....	3-29
listen.....	3-30
ntohl.....	3-30
ntohs.....	3-31
recv.....	3-32
recvfrom.....	3-33
select.....	3-35
send.....	3-37
sendto.....	3-38
setsockopt.....	3-40
shutdown.....	3-41
socket.....	3-42
Simplified Socket Library Reference.....	3-43
net_get_gateway.....	3-43
net_get_IP.....	3-43
net_get_MAC_address.....	3-43
net_get_netmask.....	3-44
tcp_close.....	3-44
tcp_connect.....	3-44

tcp_connect_nowait .....	3-45
tcp_get_remote .....	3-45
tcp_iqueue.....	3-46
tcp_listen.....	3-46
tcp_listen_nowait.....	3-47
tcp_listento .....	3-47
tcp_listento_nowait.....	3-48
tcp_ofree .....	3-48
tcp_open .....	3-48
tcp_recv .....	3-49
tcp_send.....	3-49
tcp_state.....	3-50
udp_close.....	3-50
udp_iqueue .....	3-50
udp_ofree.....	3-51
udp_open .....	3-51
udp_recv .....	3-51
udp_send.....	3-52
System Control Library Reference .....	3-52
sys_calloc .....	3-52
sys_clock_ms.....	3-52
sys_clock_s.....	3-53
sys_exit.....	3-53
sys_free.....	3-53
sys_get_info.....	3-54
sys_get_LastErrno .....	3-55
sys_get_SerialType.....	3-55
sys_getFreeMemSize.....	3-56
sys_GetServersIp.....	3-56
sys_malloc .....	3-57
sys_realloc .....	3-57
sys_restart_system .....	3-57
sys_restart_UserAP .....	3-58
sys_Set_RegisterID .....	3-58
sys_set_SerialType .....	3-58
sys_sleep_ms .....	3-59
sys_timeout.....	3-59
sysc_GetDebug.....	3-59
sysc_GetGateway .....	3-60
sysc_GetIP.....	3-60
sysc_GetIPConfig.....	3-60
sysc_GetIPLocating.....	3-61
sysc_GetName .....	3-61
sysc_GetNetmask .....	3-61
sysc_GetPassword .....	3-62
sysc_GetSerialFIFO.....	3-62
sysc_GetSerialInterface .....	3-62
sysc_GetSerialIoctl.....	3-63
sysc_SaveAndRestart .....	3-64
sysc_SetDebug.....	3-64
sysc_SetGateway .....	3-64
sysc_SetIP.....	3-65

sysc_SetIPConfig.....	3-65
sysc_SetIPLocating .....	3-65
sysc_SetName.....	3-66
sysc_SetNetmask .....	3-66
sysc_SetPassword.....	3-66
sysc_SetSerialFIFO .....	3-67
sysc_SetSerialInterface.....	3-67
sysc_SetSerialIoctl.....	3-68
sysc_SetToDefault.....	3-69
Flash ROM Access Library Reference .....	3-69
flash_erase .....	3-69
flash_length .....	3-69
flash_read .....	3-70
flash_write .....	3-70
sys_FlashErase.....	3-70
sys_FlashLength .....	3-71
sys_FlashRead .....	3-71
sys_FlashWrite .....	3-71
Debug Library Reference .....	3-72
dbg_printf .....	3-72
dbg_put_block .....	3-72
dbg_put_ch .....	3-72
dbg_put_doubleword .....	3-73
dbg_put_doubleword_hex .....	3-73
dbg_put_IP .....	3-73
dbg_put_string .....	3-74
dbg_put_word.....	3-74
dbg_put_word_hex .....	3-74
DIO Library Reference.....	3-75
DIO_ControlSingleIO.....	3-75
DIO_GetSingleIO .....	3-75
DIO_GetSingleIOStatus .....	3-75
DIO_SetSingleIO.....	3-76
Scf_getSDioMode .....	3-76
Scf_getSDioState.....	3-76
Scf_setSDioMode.....	3-77
Scf_setSDioState .....	3-77
Thread Control Library Reference.....	3-77
sys_ThreadClose.....	3-77
sys_ThreadCreate .....	3-78
sys_ThreadResume .....	3-78
sys_ThreadState .....	3-78
sys_ThreadSuspend .....	3-79
Time Server Library Reference .....	3-79
sys_GetLocalTime.....	3-79
sys_SetLocalTime .....	3-79
sysc_getTimeServer.....	3-80
sysc_getTimeZone.....	3-80
sysc_getTZoneIndex.....	3-80
sysc_setTimeServer .....	3-81
sysc_setTimeZone .....	3-81
sysc_setTZoneIndex .....	3-81

Time Zone Offsets Index.....	3-82
<b>Appendix A. External Function Calls .....</b>	<b>A-1</b>
<b>Appendix B. Service Information.....</b>	<b>B-1</b>
MOXA Internet Services .....	B-2
Problem Report Form .....	B-3
Product Return Procedure.....	B-4

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

The following topic is covered in this chapter:

 **Sample Layout**



## Sample Layout

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

<i>function name</i>	<i>brief function description</i>	<i>function type</i>
Syntax		
<pre>#include &lt;header file name&gt; function call syntax</pre>		
Arguments		
<pre>variable names          brief description of variables</pre>		
Description		
<pre>detailed function description</pre>		
Return Value		
<pre>return code #1          description of return code return code #2          description of return code</pre>		

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

<b>sio_oqueue</b>	get the length of data not yet sent out in both the system's output buffer and the driver's output buffer	<b>Port Status</b>
Syntax		
<pre>#include &lt;sdksio.h&gt; long sio_oqueue ( int port );</pre>		
Arguments		
<pre>port          async serial port number</pre>		
Description		
<pre>get the length of data not yet sent out in both the system's output buffer and the driver's output buffer</pre>		
Return Value		
<pre>&gt;= 0          length of data (bytes) still remaining driver's output buffer SIO_BADPORT   port number is invalid</pre>		

# 2

## API List

---

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

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

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

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

## Overview

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

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

## Serial I/O API

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

### Port Control

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

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

### Data Input

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

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

### Data Output

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

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

## Port Status

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

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

## Event Control

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

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

## Miscellaneous

Miscellaneous functions are special COM port functions.

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

## BSD Socket API

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

### Socket Control

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

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

### Data Input/Output

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

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

### Inquiry

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

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

## Miscellaneous

Miscellaneous functions are special socket functions.

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

## Simplified Socket API

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

### Socket Control

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

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

### Data Input/Output

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

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

### Socket Inquiry

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

Function Name	Description
<code>tcp_get_remote</code>	get connected host IP and port
<code>tcp_iqueue</code>	get length of data accumulated in TCP driver's input buffer
<code>tcp_ofree</code>	get amount of free space in TCP driver's input buffer

Function Name	Description
<code>tcp_state</code>	get TCP state
<code>udp_iqueue</code>	get length of data accumulated in UDP driver's input buffer
<code>udp_ofree</code>	get amount of free space in UDP driver's input buffer

## Port Inquiry

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

Function Name	Description
<code>net_get_gateway</code>	get local default gateway
<code>net_get_IP</code>	get local IP address
<code>net_get_MAC_address</code>	get MAC address
<code>net_get_netmask</code>	get local subnet mask

## System Control API

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

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

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

## Flash ROM Access API

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

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

## Debug API

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

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



## DIO API

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

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

## Thread Control API

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

Function Name	Description
<code>sys_ThreadClose</code>	close a thread
<code>sys_ThreadCreate</code>	create a thread
<code>sys_ThreadResume</code>	resume a thread
<code>sys_ThreadState</code>	get a thread state
<code>sys_ThreadSuspend</code>	suspend a thread

## Time Server API

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

Function Name	Description
<code>sys_GetLocalTime</code>	get local time
<code>sys_SetLocalTime</code>	set local time
<code>sysc_getTimeServer</code>	get time server
<code>sysc_getTimeZone</code>	get time zone
<code>sysc_getTZoneIndex</code>	get time zone index
<code>sysc_setTimeServer</code>	set time server
<code>sysc_setTimeZone</code>	set time zone
<code>sysc_setTZoneIndex</code>	set time zone index



### ATTENTION

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

The following topics are covered in this chapter:

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

## Serial I/O Library Reference

<b>sio_AbortRead</b>	abort when blocked from reading a block of data for <b>sio_read()</b> and <b>sio_getch()</b>	<b>Data Input</b>
<p>Syntax</p> <pre>#include &lt;sdksio.h&gt; int sio_AbortRead ( int port );</pre> <p>Arguments</p> <p><b>port</b>                      async serial port number</p> <p>Description</p> <p>abort when blocked from reading a block of data for <b>sio_read()</b> and <b>sio_getch()</b>; calling this function will cause <b>sio_read()</b> to return immediately with return code of length of data read</p> <p>Return Value</p> <p><b>SIO_OK</b>                      OK  <b>SIO_BADPORT</b>                port number is invalid</p>		
<b>sio_AbortWrite</b>	abort when blocked from writing a block of data for <b>sio_write()</b> and <b>sio_putch()</b>	<b>Data Output</b>
<p>Syntax</p> <pre>#include &lt;sdksio.h&gt; int sio_AbortWrite ( int port );</pre> <p>Arguments</p> <p><b>port</b>                      async serial port number</p> <p>Description</p> <p>abort when blocked from writing a block of data for <b>sio_write()</b> or <b>sio_putch()</b>; calling this function will cause <b>sio_write()</b> to return immediately with return code of <b>SIO_ABORT_WRITE</b></p> <p>Return Value</p> <p><b>SIO_OK</b>                      OK  <b>SIO_BADPORT</b>                port number is invalid</p>		
<b>sio_ActXoff</b>	make transmission act as if an XOFF character has been received	<b>Misc.</b>
<p>Syntax</p> <pre>#include &lt;sdksio.h&gt; int sio_ActXoff ( int port );</pre> <p>Arguments</p> <p><b>port</b>                      async serial port number</p> <p>Description</p> <p>this function causes transmission to act as if an XOFF character has been received</p> <p>Return Value</p> <p><b>SIO_OK</b>                      OK  <b>SIO_BADPORT</b>                port was not open in advance</p>		

<b>sio_ActXon</b>	make transmission act as if an XON character has been received	<b>Misc.</b>
<p>Syntax</p> <pre>#include &lt;sdksio.h&gt; int sio_ActXon ( int port );</pre> <p>Arguments</p> <p><b>port</b>                      async serial port number</p> <p>Description</p> <p>this function causes transmission to act as if an XON character has been received</p> <p>Return Value</p> <p><b>SIO_OK</b>                      OK</p> <p><b>SIO_BADPORT</b>                port number is invalid</p>		
<b>sio_baud</b>	set baud rate using the actual speed value	<b>Port Control</b>
<p>Syntax</p> <pre>#include &lt;sdksio.h&gt; int sio_baud ( int port, long speed );</pre> <p>Arguments</p> <p><b>port</b>                      async serial port number</p> <p><b>speed</b>                      true baud rate: e.g., 200, 1200, 9600, or 19200</p> <p>Description</p> <p>set baud rate using the actual speed value</p> <p>Return Value</p> <p><b>SIO_OK</b>                      OK</p> <p><b>SIO_BADPORT</b>                port number is invalid</p>		
<b>sio_break</b>	send out a BREAK signal	<b>Misc.</b>
<p>Syntax</p> <pre>#include &lt;sdksio.h&gt; int sio_break ( int port, int time );</pre> <p>Arguments</p> <p><b>port</b>                      async serial port number</p> <p><b>time</b>                      break time in tics (1/18.2 second)</p> <p>Description</p> <p>this function will block transmission until the time has expired</p> <p>Return Value</p> <p><b>SIO_OK</b>                      OK</p> <p><b>SIO_BADPORT</b>                port number is invalid</p> <p><b>SIO_BADPARAM</b>                bad parameter</p> <p><b>SIO_NOT_OPEN</b>                port was not open in advance</p>		

<b>sio_break_ex</b>	send out a BREAK signal	<b>Misc.</b>												
<p>Syntax</p> <pre>#include &lt;sdksio.h&gt; int sio_break_ex ( int port, int ms );</pre> <p>Arguments</p> <table> <tr> <td><b>port</b></td> <td>async serial port number</td> </tr> <tr> <td><b>ms</b></td> <td>break time (milliseconds)</td> </tr> </table> <p>Description</p> <p>sends out a break signal; will block transmission until time has expired; is the same as <b>sio_break()</b>, except that the time unit is measured in milliseconds</p> <p>Return Value</p> <table> <tr> <td><b>SIO_OK</b></td> <td>OK</td> </tr> <tr> <td><b>SIO_BADPORT</b></td> <td>port number is invalid</td> </tr> <tr> <td><b>SIO_BADPARM</b></td> <td>bad parameter</td> </tr> <tr> <td><b>SIO_NOT_OPEN</b></td> <td>port was not open in advance</td> </tr> </table>			<b>port</b>	async serial port number	<b>ms</b>	break time (milliseconds)	<b>SIO_OK</b>	OK	<b>SIO_BADPORT</b>	port number is invalid	<b>SIO_BADPARM</b>	bad parameter	<b>SIO_NOT_OPEN</b>	port was not open in advance
<b>port</b>	async serial port number													
<b>ms</b>	break time (milliseconds)													
<b>SIO_OK</b>	OK													
<b>SIO_BADPORT</b>	port number is invalid													
<b>SIO_BADPARM</b>	bad parameter													
<b>SIO_NOT_OPEN</b>	port was not open in advance													
<b>sio_break_irq</b>	set an event service routine to be called when a BREAK signal is received	<b>Event Control</b>												
<p>Syntax</p> <pre>#include &lt;sdksio.h&gt; int sio_break_irq ( int port, void (*func) (int port) );</pre> <p>Arguments</p> <table> <tr> <td><b>port</b></td> <td>async serial port number</td> </tr> <tr> <td><b>func</b></td> <td>event service routine entry; if <b>func</b> is NULL, this routine will be disabled</td> </tr> </table> <p>Description</p> <p>set an event service routine to be called when a BREAK signal is received; when a BREAK signal is encountered, the system will call the event service routine</p> <p>Return Value</p> <table> <tr> <td><b>SIO_OK</b></td> <td>OK</td> </tr> <tr> <td><b>SIO_BADPORT</b></td> <td>port number is invalid</td> </tr> <tr> <td><b>SIO_NOT_OPEN</b></td> <td>port was not open in advance</td> </tr> </table>			<b>port</b>	async serial port number	<b>func</b>	event service routine entry; if <b>func</b> is NULL, this routine will be disabled	<b>SIO_OK</b>	OK	<b>SIO_BADPORT</b>	port number is invalid	<b>SIO_NOT_OPEN</b>	port was not open in advance		
<b>port</b>	async serial port number													
<b>func</b>	event service routine entry; if <b>func</b> is NULL, this routine will be disabled													
<b>SIO_OK</b>	OK													
<b>SIO_BADPORT</b>	port number is invalid													
<b>SIO_NOT_OPEN</b>	port was not open in advance													
<b>sio_close</b>	disable serial port for transmitting or receiving data	<b>Port Control</b>												
<p>Syntax</p> <pre>#include &lt;sdksio.h&gt; int sio_close ( int port );</pre> <p>Arguments</p> <table> <tr> <td><b>port</b></td> <td>async serial port number</td> </tr> </table> <p>Description</p> <p>disable a serial port so that it cannot receive or transmit data</p> <p>Return Value</p> <table> <tr> <td><b>SIO_OK</b></td> <td>OK</td> </tr> <tr> <td><b>SIO_BADPORT</b></td> <td>port number is invalid</td> </tr> <tr> <td><b>SIO_NOT_OPEN</b></td> <td>port was not open in advance</td> </tr> </table>			<b>port</b>	async serial port number	<b>SIO_OK</b>	OK	<b>SIO_BADPORT</b>	port number is invalid	<b>SIO_NOT_OPEN</b>	port was not open in advance				
<b>port</b>	async serial port number													
<b>SIO_OK</b>	OK													
<b>SIO_BADPORT</b>	port number is invalid													
<b>SIO_NOT_OPEN</b>	port was not open in advance													

<code>sio_cnt_irq</code>	set an event service routine to be called when a certain amount of data has been received	Event Control												
<p>Syntax</p> <pre>#include &lt;sdksio.h&gt; int sio_cnt_irq ( int port, void (*func) (int port), int count );</pre> <p>Arguments</p> <table> <tr> <td><code>port</code></td> <td>async serial port number</td> </tr> <tr> <td><code>func</code></td> <td>event service routine entry; if <code>func</code> is NULL, this routine will be disabled</td> </tr> <tr> <td><code>count</code></td> <td>data count</td> </tr> </table> <p>Description</p> <p>set an event service routine to be called when a certain amount of data has been received; when there are <code>count</code> bytes of data received in the input buffer, the system will call the <code>func</code> service routine</p> <p>Return Value</p> <table> <tr> <td><code>SIO_OK</code></td> <td>OK</td> </tr> <tr> <td><code>SIO_BADPORT</code></td> <td>port number is invalid</td> </tr> <tr> <td><code>SIO_NOT_OPEN</code></td> <td>port was not open in advance</td> </tr> </table>			<code>port</code>	async serial port number	<code>func</code>	event service routine entry; if <code>func</code> is NULL, this routine will be disabled	<code>count</code>	data count	<code>SIO_OK</code>	OK	<code>SIO_BADPORT</code>	port number is invalid	<code>SIO_NOT_OPEN</code>	port was not open in advance
<code>port</code>	async serial port number													
<code>func</code>	event service routine entry; if <code>func</code> is NULL, this routine will be disabled													
<code>count</code>	data count													
<code>SIO_OK</code>	OK													
<code>SIO_BADPORT</code>	port number is invalid													
<code>SIO_NOT_OPEN</code>	port was not open in advance													

<code>sio_data_status</code>	check if error occurred when receiving data	Port Status								
<p>Syntax</p> <pre>#include &lt;sdksio.h&gt; int sio_data_status ( int port );</pre> <p>Arguments</p> <table> <tr> <td><code>port</code></td> <td>async serial port number</td> </tr> </table> <p>Description</p> <p>check if an error occurred when receiving data</p> <p>Return Value</p> <table> <tr> <td><code>0</code></td> <td>no error</td> </tr> <tr> <td><code>&gt;0</code></td> <td>bit 0 on: parity error bit 1 on: framing error bit 2 on: overrun error bit 3 on: overflow error</td> </tr> <tr> <td><code>SIO_BADPORT</code></td> <td>port number is invalid</td> </tr> </table>			<code>port</code>	async serial port number	<code>0</code>	no error	<code>&gt;0</code>	bit 0 on: parity error bit 1 on: framing error bit 2 on: overrun error bit 3 on: overflow error	<code>SIO_BADPORT</code>	port number is invalid
<code>port</code>	async serial port number									
<code>0</code>	no error									
<code>&gt;0</code>	bit 0 on: parity error bit 1 on: framing error bit 2 on: overrun error bit 3 on: overflow error									
<code>SIO_BADPORT</code>	port number is invalid									

<b>sio_DTR</b>	set DTR state of a port	<b>Port Control</b>								
<p>Syntax</p> <pre>#include &lt;sdksio.h&gt; int sio_DTR ( int port, int mode );</pre>										
<p>Arguments</p> <table> <tr> <td><b>port</b></td> <td>async serial port number</td> </tr> <tr> <td><b>mode</b></td> <td>0: turn DTR off 1: turn DTR on</td> </tr> </table>			<b>port</b>	async serial port number	<b>mode</b>	0: turn DTR off 1: turn DTR on				
<b>port</b>	async serial port number									
<b>mode</b>	0: turn DTR off 1: turn DTR on									
<p>Description</p> <p>set the DTR state of a port</p>										
<p>Return Value</p> <table> <tr> <td><b>SIO_OK</b></td> <td>OK</td> </tr> <tr> <td><b>SIO_BADPORT</b></td> <td>port number is invalid</td> </tr> <tr> <td><b>SIO_BADPARM</b></td> <td>bad parameter</td> </tr> <tr> <td><b>SIO_NOT_OPEN</b></td> <td>port was not open in advance</td> </tr> </table>			<b>SIO_OK</b>	OK	<b>SIO_BADPORT</b>	port number is invalid	<b>SIO_BADPARM</b>	bad parameter	<b>SIO_NOT_OPEN</b>	port was not open in advance
<b>SIO_OK</b>	OK									
<b>SIO_BADPORT</b>	port number is invalid									
<b>SIO_BADPARM</b>	bad parameter									
<b>SIO_NOT_OPEN</b>	port was not open in advance									

<b>sio_flowctrl</b>	set hardware and/or software flow control	<b>Port Control</b>								
<p>Syntax</p> <pre>#include &lt;sdksio.h&gt; int sio_flowctrl ( int port, int mode );</pre>										
<p>Arguments</p> <table> <tr> <td><b>port</b></td> <td>async serial port number</td> </tr> <tr> <td><b>mode</b></td> <td>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)</td> </tr> </table>			<b>port</b>	async serial port number	<b>mode</b>	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)				
<b>port</b>	async serial port number									
<b>mode</b>	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)									
<p>Description</p> <p>set the hardware and/or software flow control</p>										
<p>Return Value</p> <table> <tr> <td><b>SIO_OK</b></td> <td>OK</td> </tr> <tr> <td><b>SIO_BADPORT</b></td> <td>port number is invalid</td> </tr> <tr> <td><b>SIO_BADPARM</b></td> <td>bad parameter</td> </tr> <tr> <td><b>SIO_NOT_OPEN</b></td> <td>port was not open in advance</td> </tr> </table>			<b>SIO_OK</b>	OK	<b>SIO_BADPORT</b>	port number is invalid	<b>SIO_BADPARM</b>	bad parameter	<b>SIO_NOT_OPEN</b>	port was not open in advance
<b>SIO_OK</b>	OK									
<b>SIO_BADPORT</b>	port number is invalid									
<b>SIO_BADPARM</b>	bad parameter									
<b>SIO_NOT_OPEN</b>	port was not open in advance									

<b>sio_flush</b>	flush the driver's input/output buffer	<b>Port Control</b>						
Syntax <pre>#include &lt;sdksio.h&gt; int sio_flush ( int port, int func );</pre>								
Arguments <table> <tr> <td><b>port</b></td> <td>async serial port number</td> </tr> <tr> <td><b>func</b></td> <td>flush action 0: flush input buffer 1: flush output buffer 2: flush input &amp; output buffer</td> </tr> </table>			<b>port</b>	async serial port number	<b>func</b>	flush action 0: flush input buffer 1: flush output buffer 2: flush input & output buffer		
<b>port</b>	async serial port number							
<b>func</b>	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 <table> <tr> <td><b>SIO_OK</b></td> <td>OK</td> </tr> <tr> <td><b>SIO_BADPORT</b></td> <td>port number is invalid</td> </tr> <tr> <td><b>SIO_BADPARM</b></td> <td>bad parameter</td> </tr> </table>			<b>SIO_OK</b>	OK	<b>SIO_BADPORT</b>	port number is invalid	<b>SIO_BADPARM</b>	bad parameter
<b>SIO_OK</b>	OK							
<b>SIO_BADPORT</b>	port number is invalid							
<b>SIO_BADPARM</b>	bad parameter							

<b>sio_getbaud</b>	get serial port's baud rate setting	<b>Port Status</b>				
Syntax <pre>#include &lt;sdksio.h&gt; long sio_getbaud ( int port );</pre>						
Arguments <table> <tr> <td><b>port</b></td> <td>async serial port number</td> </tr> </table>			<b>port</b>	async serial port number		
<b>port</b>	async serial port number					
Description get the serial port's baud rate setting; the return value is the actual baud rate; a return value of 9600 means 9600 bps, and a return value of 200 means 200 bps						
Return Value <table> <tr> <td>&gt; 0</td> <td>baud rate</td> </tr> <tr> <td><b>SIO_BADPORT</b></td> <td>port number is invalid</td> </tr> </table>			> 0	baud rate	<b>SIO_BADPORT</b>	port number is invalid
> 0	baud rate					
<b>SIO_BADPORT</b>	port number is invalid					

<b>sio_getch</b>	read one character from driver's input buffer	<b>Data Input</b>										
Syntax <pre>#include &lt;sdksio.h&gt; int sio_getch ( int port );</pre>												
Arguments <table> <tr> <td><b>port</b></td> <td>async serial port number</td> </tr> </table>			<b>port</b>	async serial port number								
<b>port</b>	async serial port number											
Description read one character from the driver's input buffer												
Return Value <table> <tr> <td>0 to 255</td> <td>ASCII code of the character received</td> </tr> <tr> <td><b>SIO_BADPORT</b></td> <td>port number is invalid</td> </tr> <tr> <td><b>SIO_NODATA</b></td> <td>no data to read</td> </tr> <tr> <td><b>SIO_BADPARM</b></td> <td>bad parameter</td> </tr> <tr> <td><b>SIO_NOT_OPEN</b></td> <td>port was not open in advance</td> </tr> </table>			0 to 255	ASCII code of the character received	<b>SIO_BADPORT</b>	port number is invalid	<b>SIO_NODATA</b>	no data to read	<b>SIO_BADPARM</b>	bad parameter	<b>SIO_NOT_OPEN</b>	port was not open in advance
0 to 255	ASCII code of the character received											
<b>SIO_BADPORT</b>	port number is invalid											
<b>SIO_NODATA</b>	no data to read											
<b>SIO_BADPARM</b>	bad parameter											
<b>SIO_NOT_OPEN</b>	port was not open in advance											



<b>sio_getflow</b>	get serial port's hardware and software flow control settings	<b>Port Status</b>
<p>Syntax</p> <pre>#include &lt;sdksio.h&gt; int sio_getflow ( int port );</pre> <p>Arguments</p> <p><b>port</b>                      async serial port number</p> <p>Description</p> <p>get the serial port's hardware and software flow control settings; refer to <b>sio_flowctrl()</b> for detail</p> <p>Return Value</p> <p><b>&gt;=0</b>                      bit 0: 1 CTS flow control  bit 1: RTS flow control  bit 2: Tx XON/XOFF flow control  bit 3: Rx XON/XOFF flow control</p> <p><b>SIO_BADPORT</b>              port number is invalid</p>		
<b>sio_getmode</b>	get serial port's mode settings	<b>Port Status</b>
<p>Syntax</p> <pre>#include &lt;sdksio.h&gt; int sio_getmode ( int port );</pre> <p>Arguments</p> <p><b>port</b>                      async serial port number</p> <p>Description</p> <p>get the serial port's mode settings; refer to the description of <b>sio_ioctl()</b> to see the mode settings</p> <p>Return Value</p> <p><b>&gt;=0</b>                      the mode settings, see <b>sio_ioctl()</b> for detail</p> <p><b>SIO_BADPORT</b>              port number is invalid</p>		
<b>sio_GetReadTimeouts</b>	get timeout values for <b>sio_read()</b> and <b>sio_getch()</b>	<b>Data Input</b>
<p>Syntax</p> <pre>#include &lt;sdksio.h&gt; int sio_GetReadTimeouts ( int port, DWORD *TotalTimeouts, DWORD *IntervalTimeouts );</pre> <p>Arguments</p> <p><b>port</b>                      async serial port number</p> <p><b>TotalTimeouts</b>            a pointer to buffer to retrieve total timeout value</p> <p><b>IntervalTimeouts</b>        a pointer to buffer to retrieve interval timeout value</p> <p>Description</p> <p>get timeout values for <b>sio_read()</b> and <b>sio_getch()</b></p> <p>Return Value</p> <p><b>SIO_OK</b>                      OK</p> <p><b>SIO_BADPORT</b>              port number is invalid</p>		

<b>sio_GetWriteTimeouts</b>	get timeout value for <b>sio_write()</b> and <b>sio_putch()</b>	<b>Data Output</b>
Syntax		
<pre>#include &lt;sdksio.h&gt; int sio_GetWriteTimeouts ( int port, DWORD *TotalTimeouts );</pre>		
Arguments		
<b>port</b>	async serial port number	
<b>TotalTimeouts</b>	a pointer to buffer to retrieve the total timeout value	
Description		
get timeout values for <b>sio_write()</b> and <b>sio_putch()</b>		
Return Value		
<b>SIO_OK</b>	OK	
<b>SIO_BADPORT</b>	port number is invalid	

<code>sio_ioctl</code>	modify settings of serial port's I/O control register	Port Control
Syntax		
<pre>#include &lt;sdksio.h&gt; int sio_ioctl ( int port, int baud, int mode );</pre>		
Arguments		
<code>port</code>	async serial port number	
<code>baud</code>	(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
<code>mode</code>	<b>bit_cnt</b> OR <b>stop_bit</b> OR <b>parity</b>	
	<b>bit_cnt</b> (bits 0-1)	
	0x00: data bit 5	
	0x01: data bit 6	
	0x02: data bit 7	
	0x03: data bit 8	
	<b>stop_bit</b> (bit 2)	
	0x00: stop bit 1	
	0x04: stop bit 1.5 or 2	
	<b>parity</b> (bits 3- 5)	
	0x00: no parity	
	0x08: odd parity	
	0x18: even parity	
	0x28: mark parity	
	0x38: space parity	
Description		
modify the settings of the serial port's I/O control register, such as baud rate, parity, data bits, and stop bit		
Return Value		
<code>SIO_OK</code>	OK	
<code>SIO_BADPORT</code>	port number is invalid	
<code>SIO_BADPARM</code>	bad parameter	
<code>SIO_NOT_OPEN</code>	port was not open in advance	

<code>sio_iqueue</code>	get length of data accumulated in system and driver input buffers	Port Status
<p>Syntax</p> <pre>#include &lt;sdksio.h&gt; long sio_iqueue ( int port );</pre> <p>Arguments</p> <p><code>port</code>                      async serial port number</p> <p>Description</p> <p>get the length of data accumulated in the system's input buffer and driver's input buffer; note that even when <code>sio_iqueue()</code> returns a zero value, there may be a few characters remaining in the RS-232 UART chip that are not yet known</p> <p>Return Value</p> <p><code>&gt;= 0</code>                      data currently in input buffer (bytes)</p> <p><code>SIO_BADPORT</code>              port number is invalid</p>		

<code>sio_lctrl</code>	set DTR and RTS states	Port Control
<p>Syntax</p> <pre>#include &lt;sdksio.h&gt; int sio_lctrl ( int port, int mode );</pre> <p>Arguments</p> <p><code>port</code>                      async serial port number</p> <p><code>mode</code>                      <code>C_DTR</code> (bit 0)</p> <p>                            <code>C_RTS</code> (bit 1)</p> <p>Description</p> <p>set both the DTR and RTS states</p> <p>Return Value</p> <p><code>SIO_OK</code>                      OK</p> <p><code>SIO_BADPORT</code>              port number is invalid</p> <p><code>SIO_BADPARAM</code>            bad parameter</p> <p><code>SIO_NOT_OPEN</code>            port was not open in advance</p> <p><code>SIO_RTS_BY_HW</code>           cannot control the port because it is set as auto hardware flow control by <code>sio_flowctrl()</code></p>		

<code>sio_linput</code>	read a block of data ending with termination character	Data Input																		
<p>Syntax</p> <pre>#include &lt;sdksio.h&gt; int sio_linput ( int port, char *buf, int len, int term );</pre> <p>Arguments</p> <table> <tr> <td><code>port</code></td> <td>async serial port number</td> </tr> <tr> <td><code>buf</code></td> <td>receive buffer pointer</td> </tr> <tr> <td><code>len</code></td> <td>buffer length (bytes)</td> </tr> <tr> <td><code>term</code></td> <td>terminator code</td> </tr> </table> <p>Description</p> <p>read a block of data from the driver's input buffer until the terminator character is encountered or <code>len</code> bytes of data are read</p> <p>Return Value</p> <table> <tr> <td><code>&gt; 0</code></td> <td>length of data received (bytes)</td> </tr> <tr> <td><code>= 0</code></td> <td>no data received</td> </tr> <tr> <td><code>SIO_BADPORT</code></td> <td>port number is invalid</td> </tr> <tr> <td><code>SIO_BADPARM</code></td> <td>bad parameter</td> </tr> <tr> <td><code>SIO_NOT_OPEN</code></td> <td>port was not open in advance</td> </tr> </table>			<code>port</code>	async serial port number	<code>buf</code>	receive buffer pointer	<code>len</code>	buffer length (bytes)	<code>term</code>	terminator code	<code>&gt; 0</code>	length of data received (bytes)	<code>= 0</code>	no data received	<code>SIO_BADPORT</code>	port number is invalid	<code>SIO_BADPARM</code>	bad parameter	<code>SIO_NOT_OPEN</code>	port was not open in advance
<code>port</code>	async serial port number																			
<code>buf</code>	receive buffer pointer																			
<code>len</code>	buffer length (bytes)																			
<code>term</code>	terminator code																			
<code>&gt; 0</code>	length of data received (bytes)																			
<code>= 0</code>	no data received																			
<code>SIO_BADPORT</code>	port number is invalid																			
<code>SIO_BADPARM</code>	bad parameter																			
<code>SIO_NOT_OPEN</code>	port was not open in advance																			

<code>sio_lstatus</code>	get status of the serial line	Port Status														
<p>Syntax</p> <pre>#include &lt;sdksio.h&gt; int sio_lstatus ( int port );</pre> <p>Arguments</p> <table> <tr> <td><code>port</code></td> <td>async serial port number</td> </tr> </table> <p>Description</p> <p>get the status of the line</p> <p>Return Value</p> <table> <tr> <td><code>&gt;= 0</code></td> <td>current line status</td> </tr> <tr> <td></td> <td>bit 0: <code>S_CTS</code></td> </tr> <tr> <td></td> <td>bit 1: <code>S_DSR</code></td> </tr> <tr> <td></td> <td>bit 2: <code>S_RI</code></td> </tr> <tr> <td></td> <td>bit 3: <code>S_CD</code></td> </tr> <tr> <td><code>SIO_BADPORT</code></td> <td>port number is invalid</td> </tr> </table>			<code>port</code>	async serial port number	<code>&gt;= 0</code>	current line status		bit 0: <code>S_CTS</code>		bit 1: <code>S_DSR</code>		bit 2: <code>S_RI</code>		bit 3: <code>S_CD</code>	<code>SIO_BADPORT</code>	port number is invalid
<code>port</code>	async serial port number															
<code>&gt;= 0</code>	current line status															
	bit 0: <code>S_CTS</code>															
	bit 1: <code>S_DSR</code>															
	bit 2: <code>S_RI</code>															
	bit 3: <code>S_CD</code>															
<code>SIO_BADPORT</code>	port number is invalid															

<b>sio_modem_irq</b>	set event service routine to be called when line status has changed	<b>Event Control</b>
<p>Syntax</p> <pre>#include &lt;sdksio.h&gt; int sio_modem_irq ( int port, void (*func) (int port) );</pre> <p>Arguments</p> <p><b>port</b>                      async serial port number</p> <p><b>func</b>                      event service routine entry; if the <b>func</b> is NULL, it will disable this routine</p> <p>Description</p> <p>set event service routine to be called when line status has changed; when line status (CTS, DSR, CD, RI) changes, the system will call the event service routine</p> <p>Return Value</p> <p><b>SIO_OK</b>                      OK</p> <p><b>SIO_BADPORT</b>              port number is invalid</p> <p><b>SIO_NOT_OPEN</b>            port was not open in advance</p>		
<b>sio_ofree</b>	get amount of free space in driver's output buffer	<b>Port Status</b>
<p>Syntax</p> <pre>#include &lt;sdksio.h&gt; long sio_ofree ( int port );</pre> <p>Arguments</p> <p><b>port</b>                      async serial port number</p> <p>Description</p> <p>get the amount of free space in the driver's output buffer</p> <p>Return Value</p> <p><b>&gt;= 0</b>                      amount of free space in output buffer (bytes)</p> <p><b>SIO_BADPORT</b>              port number is invalid</p>		
<b>sio_open</b>	enable serial port to transmit and receive data	<b>Port Control</b>
<p>Syntax</p> <pre>#include &lt;sdksio.h&gt; int sio_open ( int port );</pre> <p>Arguments</p> <p><b>port</b>                      async serial port number</p> <p>Description</p> <p>enable a serial port to transmit and receive data; after calling <b>sio_open()</b>, the initial status of this serial port is the same as the last setting or configuration setting</p> <p>Return Value</p> <p><b>&gt;= 0</b>                      indicates successful open action, and return value is a descriptor referencing the port; programmer can use this descriptor in the <b>select()</b> function (from the socket API group) to carry out a data read/write operation</p> <p><b>SIO_BADPORT</b>              port number is invalid</p>		

<b>sio_oqueue</b>	get the amount of data remaining in system and driver output buffers	<b>Port Status</b>				
Syntax <pre>#include &lt;sdksio.h&gt; long sio_oqueue ( int port );</pre>						
Arguments <table> <tr> <td><b>port</b></td> <td>async serial port number</td> </tr> </table>			<b>port</b>	async serial port number		
<b>port</b>	async serial port number					
Description get the amount of data not yet sent out in both the system's output buffer and the driver's output buffer						
Return Value <table> <tr> <td><b>&gt;= 0</b></td> <td>amount of data (bytes), still remaining in driver's output buffer</td> </tr> <tr> <td><b>SIO_BADPORT</b></td> <td>port number is invalid</td> </tr> </table>			<b>&gt;= 0</b>	amount of data (bytes), still remaining in driver's output buffer	<b>SIO_BADPORT</b>	port number is invalid
<b>&gt;= 0</b>	amount of data (bytes), still remaining in driver's output buffer					
<b>SIO_BADPORT</b>	port number is invalid					

<b>sio_putch</b>	write a character into driver's output buffer	<b>Data Output</b>												
Syntax <pre>#include &lt;sdksio.h&gt; int sio_putch ( int port, int term );</pre>														
Arguments <table> <tr> <td><b>port</b></td> <td>async serial port number</td> </tr> <tr> <td><b>term</b></td> <td>the character to be written</td> </tr> </table>			<b>port</b>	async serial port number	<b>term</b>	the character to be written								
<b>port</b>	async serial port number													
<b>term</b>	the character to be written													
Description write a character into driver's output buffer														
Return Value <table> <tr> <td><b>SIO_OK</b></td> <td>OK</td> </tr> <tr> <td><b>SIO_BADPORT</b></td> <td>port number is invalid</td> </tr> <tr> <td><b>SIO_BADPARG</b></td> <td>bad parameter</td> </tr> <tr> <td><b>SIO_ABORT_WRITE</b></td> <td>user abort blocked write</td> </tr> <tr> <td><b>SIO_WRITETIMEOUT</b></td> <td>write timeout has occurred</td> </tr> <tr> <td><b>SIO_NOT_OPEN</b></td> <td>port was not open in advance</td> </tr> </table>			<b>SIO_OK</b>	OK	<b>SIO_BADPORT</b>	port number is invalid	<b>SIO_BADPARG</b>	bad parameter	<b>SIO_ABORT_WRITE</b>	user abort blocked write	<b>SIO_WRITETIMEOUT</b>	write timeout has occurred	<b>SIO_NOT_OPEN</b>	port was not open in advance
<b>SIO_OK</b>	OK													
<b>SIO_BADPORT</b>	port number is invalid													
<b>SIO_BADPARG</b>	bad parameter													
<b>SIO_ABORT_WRITE</b>	user abort blocked write													
<b>SIO_WRITETIMEOUT</b>	write timeout has occurred													
<b>SIO_NOT_OPEN</b>	port was not open in advance													

<b>sio_read</b>	read data from the driver's input buffer	<b>Data Input</b>										
<p>Syntax</p> <pre>#include &lt;sdksio.h&gt; int sio_read ( int port, char *buf, int len );</pre>												
<p>Arguments</p> <table style="width: 100%; border: none;"> <tr> <td style="width: 15%;"><b>port</b></td> <td>async serial port number</td> </tr> <tr> <td><b>buf</b></td> <td>receive buffer pointer</td> </tr> <tr> <td><b>len</b></td> <td>buffer length (bytes)</td> </tr> </table>			<b>port</b>	async serial port number	<b>buf</b>	receive buffer pointer	<b>len</b>	buffer length (bytes)				
<b>port</b>	async serial port number											
<b>buf</b>	receive buffer pointer											
<b>len</b>	buffer length (bytes)											
<p>Description</p> <p><b>sio_read()</b> reads data from the driver's input buffer. If the user's buffer is large enough to hold the data in the driver's input buffer, then the entire contents of the driver's input buffer will be transferred to the user's buffer. Otherwise, only <b>len</b> bytes will be transferred to the user's buffer.</p> <p><b>sio_SetReadTimeout()</b> can be used to set timeouts for <b>sio_read()</b>.  <b>sio_AbortRead()</b> can be used to abort any blocked <b>sio_read()</b>.</p>												
<p>Return Value</p> <table style="width: 100%; border: none;"> <tr> <td style="width: 15%;">&gt; 0</td> <td>length of data received (bytes)</td> </tr> <tr> <td>= 0</td> <td>no data received</td> </tr> <tr> <td><b>SIO_BADPORT</b></td> <td>port number is invalid</td> </tr> <tr> <td><b>SIO_BADPARM</b></td> <td>bad parameter</td> </tr> <tr> <td><b>SIO_NOT_OPEN</b></td> <td>port was not open in advance</td> </tr> </table>			> 0	length of data received (bytes)	= 0	no data received	<b>SIO_BADPORT</b>	port number is invalid	<b>SIO_BADPARM</b>	bad parameter	<b>SIO_NOT_OPEN</b>	port was not open in advance
> 0	length of data received (bytes)											
= 0	no data received											
<b>SIO_BADPORT</b>	port number is invalid											
<b>SIO_BADPARM</b>	bad parameter											
<b>SIO_NOT_OPEN</b>	port was not open in advance											

<b>sio_RTS</b>	set RTS state of port	<b>Port Control</b>										
<p>Syntax</p> <pre>#include &lt;sdksio.h&gt; int sio_RTS ( int port, int mode );</pre>												
<p>Arguments</p> <table style="width: 100%; border: none;"> <tr> <td style="width: 15%;"><b>port</b></td> <td>async serial port number</td> </tr> <tr> <td><b>mode</b></td> <td>0: turn RTS off 1: turn RTS on</td> </tr> </table>			<b>port</b>	async serial port number	<b>mode</b>	0: turn RTS off 1: turn RTS on						
<b>port</b>	async serial port number											
<b>mode</b>	0: turn RTS off 1: turn RTS on											
<p>Description</p> <p>set RTS state of port</p>												
<p>Return Value</p> <table style="width: 100%; border: none;"> <tr> <td style="width: 15%;"><b>SIO_OK</b></td> <td>OK</td> </tr> <tr> <td><b>SIO_BADPORT</b></td> <td>port number is invalid</td> </tr> <tr> <td><b>SIO_BADPARM</b></td> <td>bad parameter</td> </tr> <tr> <td><b>SIO_RTS_BY_HW</b></td> <td>cannot control the port because it is set as auto H/W flow control by <b>sio_flowctrl()</b></td> </tr> <tr> <td><b>SIO_NOT_OPEN</b></td> <td>port was not open in advance</td> </tr> </table>			<b>SIO_OK</b>	OK	<b>SIO_BADPORT</b>	port number is invalid	<b>SIO_BADPARM</b>	bad parameter	<b>SIO_RTS_BY_HW</b>	cannot control the port because it is set as auto H/W flow control by <b>sio_flowctrl()</b>	<b>SIO_NOT_OPEN</b>	port was not open in advance
<b>SIO_OK</b>	OK											
<b>SIO_BADPORT</b>	port number is invalid											
<b>SIO_BADPARM</b>	bad parameter											
<b>SIO_RTS_BY_HW</b>	cannot control the port because it is set as auto H/W flow control by <b>sio_flowctrl()</b>											
<b>SIO_NOT_OPEN</b>	port was not open in advance											



<b>sio_SetReadTimeouts</b>	set timeout values for <b>sio_read()</b> and <b>sio_getch()</b>	<b>Data Input</b>										
<p>Syntax</p> <pre>#include &lt;sdksio.h&gt; int sio_SetReadTimeouts ( int port, DWORD TotalTimeouts, DWORD IntervalTimeouts );</pre> <p>Arguments</p> <table> <tr> <td><b>port</b></td> <td>async serial port number</td> </tr> <tr> <td><b>TotalTimeouts</b></td> <td>total timeout values (milliseconds)</td> </tr> <tr> <td><b>IntervalTimeouts</b></td> <td>interval timeout values (milliseconds)</td> </tr> </table> <p>Description</p> <p>set timeout values for <b>sio_read()</b> and <b>sio_getch()</b>; the default <b>TotalTimeouts</b> value is 0xFFFFFFFF and the default <b>IntervalTimeouts</b> value is 0, which enables <b>sio_read()</b> to return immediately</p> <p>Return Value</p> <table> <tr> <td><b>SIO_OK</b></td> <td>OK</td> </tr> <tr> <td><b>SIO_BADPORT</b></td> <td>port number is invalid</td> </tr> </table>			<b>port</b>	async serial port number	<b>TotalTimeouts</b>	total timeout values (milliseconds)	<b>IntervalTimeouts</b>	interval timeout values (milliseconds)	<b>SIO_OK</b>	OK	<b>SIO_BADPORT</b>	port number is invalid
<b>port</b>	async serial port number											
<b>TotalTimeouts</b>	total timeout values (milliseconds)											
<b>IntervalTimeouts</b>	interval timeout values (milliseconds)											
<b>SIO_OK</b>	OK											
<b>SIO_BADPORT</b>	port number is invalid											
<b>sio_SetWriteTimeouts</b>	set timeout value for <b>sio_write()</b> and <b>sio_putch()</b>	<b>Data Output</b>										
<p>Syntax</p> <pre>#include &lt;sdksio.h&gt; int sio_SetWriteTimeouts ( int port, DWORD TotalTimeouts );</pre> <p>Arguments</p> <table> <tr> <td><b>port</b></td> <td>async serial port number</td> </tr> <tr> <td><b>TotalTimeouts</b></td> <td>total timeout value (milliseconds)</td> </tr> </table> <p>Description</p> <p>set timeout value for <b>sio_write()</b> and <b>sio_putch()</b>; the default value of write timeout is 0xFFFFFFFF, which enables <b>sio_write()</b> and <b>sio_putch()</b> to return immediately without blocking at all</p> <p>the value 0 enables <b>sio_write()</b> to always block until finished writing data</p> <p>Return Value</p> <table> <tr> <td><b>SIO_OK</b></td> <td>OK</td> </tr> <tr> <td><b>SIO_BADPORT</b></td> <td>port number is invalid</td> </tr> <tr> <td><b>SIO_BADPARAM</b></td> <td>bad parameter</td> </tr> </table>			<b>port</b>	async serial port number	<b>TotalTimeouts</b>	total timeout value (milliseconds)	<b>SIO_OK</b>	OK	<b>SIO_BADPORT</b>	port number is invalid	<b>SIO_BADPARAM</b>	bad parameter
<b>port</b>	async serial port number											
<b>TotalTimeouts</b>	total timeout value (milliseconds)											
<b>SIO_OK</b>	OK											
<b>SIO_BADPORT</b>	port number is invalid											
<b>SIO_BADPARAM</b>	bad parameter											

<b>sio_term_irq</b>	set an event service routine to be called when the terminator character is received	<b>Event Control</b>												
<p>Syntax</p> <pre>#include &lt;sdksio.h&gt; int sio_term_irq ( int port, void (*func) (int port), char code );</pre> <p>Arguments</p> <table> <tr> <td><b>port</b></td> <td>async serial port number</td> </tr> <tr> <td><b>func</b></td> <td>event service routine entry; if the <b>func</b> is NULL, it will disable this routine</td> </tr> <tr> <td><b>code</b></td> <td>terminator character code</td> </tr> </table> <p>Description</p> <p>set an event service routine to be called when the terminator character is received; when the terminator character is received, the system will call the event service routine</p> <p>Return Value</p> <table> <tr> <td><b>SIO_OK</b></td> <td>OK</td> </tr> <tr> <td><b>SIO_BADPORT</b></td> <td>port number is invalid</td> </tr> <tr> <td><b>SIO_NOT_OPEN</b></td> <td>port was not open in advance</td> </tr> </table>			<b>port</b>	async serial port number	<b>func</b>	event service routine entry; if the <b>func</b> is NULL, it will disable this routine	<b>code</b>	terminator character code	<b>SIO_OK</b>	OK	<b>SIO_BADPORT</b>	port number is invalid	<b>SIO_NOT_OPEN</b>	port was not open in advance
<b>port</b>	async serial port number													
<b>func</b>	event service routine entry; if the <b>func</b> is NULL, it will disable this routine													
<b>code</b>	terminator character code													
<b>SIO_OK</b>	OK													
<b>SIO_BADPORT</b>	port number is invalid													
<b>SIO_NOT_OPEN</b>	port was not open in advance													
<b>sio_Tx_empty_irq</b>	set an event service routine to be called when output buffer is cleared	<b>Event Control</b>												
<p>Syntax</p> <pre>#include &lt;sdksio.h&gt; int sio_Tx_empty_irq ( int port, void (*func) (int port) );</pre> <p>Arguments</p> <table> <tr> <td><b>port</b></td> <td>async serial port number</td> </tr> <tr> <td><b>func</b></td> <td>event service routine entry; if the <b>func</b> is NULL, it will disable this routine</td> </tr> </table> <p>Description</p> <p>set an event service routine to be called when last character in output buffer is sent; when the Tx empty signal is encountered, the system will call the event service routine</p> <p>Return Value</p> <table> <tr> <td><b>SIO_OK</b></td> <td>OK</td> </tr> <tr> <td><b>SIO_BADPORT</b></td> <td>port number is invalid</td> </tr> <tr> <td><b>SIO_NOT_OPEN</b></td> <td>port was not open in advance</td> </tr> </table>			<b>port</b>	async serial port number	<b>func</b>	event service routine entry; if the <b>func</b> is NULL, it will disable this routine	<b>SIO_OK</b>	OK	<b>SIO_BADPORT</b>	port number is invalid	<b>SIO_NOT_OPEN</b>	port was not open in advance		
<b>port</b>	async serial port number													
<b>func</b>	event service routine entry; if the <b>func</b> is NULL, it will disable this routine													
<b>SIO_OK</b>	OK													
<b>SIO_BADPORT</b>	port number is invalid													
<b>SIO_NOT_OPEN</b>	port was not open in advance													

<code>sio_Tx_hold</code>	check why data could not be transmitted	Port Status
Syntax		
<pre>#include &lt;sdksio.h&gt; int sio_Tx_hold ( int port );</pre>		
Arguments		
<code>port</code>	async serial port number	
Description		
check the reason why data could not be transmitted		
Return Value		
<code>&gt;=0</code>	bit 0 on: data could not transmitted because CTS is low bit 1 on: data could not transmitted because XOFF char received	
<code>SIO_BADPORT</code>	port number is invalid	

<code>sio_write</code>	write a block of data to driver's output buffer	Data Output
Syntax		
<pre>#include &lt;sdksio.h&gt; int sio_write ( int port, char *buf, int len );</pre>		
Arguments		
<code>port</code>	async serial port number	
<code>buf</code>	transmit string pointer	
<code>len</code>	transmit string length (bytes)	
Description		
<p><code>sio_write</code> writes a block of data to the driver's output buffer. The actual length of data written depends on the amount of free space in the driver's output buffer. <code>sio_write()</code> is always non-block by default.</p> <p>Use <code>sio_SetWriteTimeout()</code> to set the timeout for <code>sio_write()</code>. <code>SIO_WRITETIMEOUT</code> will be returned from <code>sio_write()</code> when the write function times out.</p> <p><code>sio_AbortWrite()</code> can be used to abort any blocked <code>sio_write()</code> with return value <code>SIO_ABORT_WRITE</code>.</p>		
Return Value		
<code>&gt;= 0</code>	length of data transmitted (bytes)	
<code>SIO_BADPORT</code>	port number is invalid	
<code>SIO_BADPARAM</code>	bad parameter	
<code>SIO_ABORT_WRITE</code>	user abort blocked write	
<code>SIO_WRITETIMEOUT</code>	write timeout has occurred	
<code>SIO_NOT_OPEN</code>	port was not open in advance	

## BSD Socket Library Reference

<b>accept</b>	accept a connection on a socket	Socket Control										
<p>Syntax</p> <pre>#include &lt;sdksock.h&gt; int accept ( int s, SOCKADDR *addr, int *addrlen );</pre>												
<p>Arguments</p> <table> <tr> <td><b>s</b></td> <td>a descriptor identifying a socket which is listening for connections after a <b>listen()</b></td> </tr> <tr> <td><b>addr</b></td> <td>an optional pointer to a buffer that receives the address of the connecting entity, as known to the communications layer; exact format of the <b>addr</b> argument is determined by the address family established when the socket was created</td> </tr> <tr> <td><b>addrlen</b></td> <td>an optional pointer to an integer that contains the length of the address <b>addr</b></td> </tr> </table>			<b>s</b>	a descriptor identifying a socket which is listening for connections after a <b>listen()</b>	<b>addr</b>	an optional pointer to a buffer that receives the address of the connecting entity, as known to the communications layer; exact format of the <b>addr</b> argument is determined by the address family established when the socket was created	<b>addrlen</b>	an optional pointer to an integer that contains the length of the address <b>addr</b>				
<b>s</b>	a descriptor identifying a socket which is listening for connections after a <b>listen()</b>											
<b>addr</b>	an optional pointer to a buffer that receives the address of the connecting entity, as known to the communications layer; exact format of the <b>addr</b> argument is determined by the address family established when the socket was created											
<b>addrlen</b>	an optional pointer to an integer that contains the length of the address <b>addr</b>											
<p>Description</p> <p>This routine extracts the first connection on the queue of pending connections on <b>s</b>, creates a new socket with the same properties as <b>s</b>, 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, <b>accept()</b> blocks the caller until a connection is present. If the socket is marked non-blocking and no pending connections are present on the queue, <b>accept()</b> returns an error as described below. The accepted socket may not be used to accept more connections. The original socket remains open.</p> <p>The argument <b>addr</b> 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 <b>addr</b> parameter is determined by the address family in which the communication is occurring. The <b>addrlen</b> is a value-result parameter; it should initially contain the amount of space pointed to by <b>addr</b>. On return, it will contain the actual length (bytes), of the address returned. This call is used with connection-based socket types such as <b>SOCK_STREAM</b>. If <b>addr</b> or <b>addrlen</b> are equal to <b>NULL</b>, then no information about the remote address of the accepted socket is returned.</p>												
<p>Return Value</p> <p>If there are no errors, <b>accept()</b> returns the descriptor for the accepted packet. Otherwise, a value of -1 is returned, and the global variable <b>errno</b> will contain one of the following values.</p>												
<p>Error Codes</p> <table> <tr> <td><b>EBADF</b></td> <td>the first argument does not specify a valid descriptor</td> </tr> <tr> <td><b>EOPNOTSUPP</b></td> <td>the socket is not of type <b>SOCK_STREAM</b></td> </tr> <tr> <td><b>EFAULT</b></td> <td>the pointer in one of the arguments is invalid</td> </tr> <tr> <td><b>EWOULDBLOCK</b></td> <td>the socket is marked non-blocking and no connections are waiting to be accepted</td> </tr> <tr> <td><b>EFFILE</b></td> <td>the initial system file table is full</td> </tr> </table>			<b>EBADF</b>	the first argument does not specify a valid descriptor	<b>EOPNOTSUPP</b>	the socket is not of type <b>SOCK_STREAM</b>	<b>EFAULT</b>	the pointer in one of the arguments is invalid	<b>EWOULDBLOCK</b>	the socket is marked non-blocking and no connections are waiting to be accepted	<b>EFFILE</b>	the initial system file table is full
<b>EBADF</b>	the first argument does not specify a valid descriptor											
<b>EOPNOTSUPP</b>	the socket is not of type <b>SOCK_STREAM</b>											
<b>EFAULT</b>	the pointer in one of the arguments is invalid											
<b>EWOULDBLOCK</b>	the socket is marked non-blocking and no connections are waiting to be accepted											
<b>EFFILE</b>	the initial system file table is full											

<b>bind</b>	associate a local address with a socket	Socket Control
Syntax		
<pre>#include &lt;sdksoc.h&gt; int bind ( int s, SOCKADDR *name, int namelen );</pre>		
Arguments		
<b>s</b>	a descriptor identifying an unbound socket	
<b>name</b>	the address to assign to the socket	
<b>namelen</b>	length of the value in <b>name</b>	
Description		
<p>This routine is used on an unconnected datagram or stream socket, before subsequent <b>connect()</b> or <b>listen()</b> routines. When a socket is created with <b>socket()</b>, it exists in a name space (address family), but it has no name assigned. <b>bind()</b> establishes the local association (host address/port number) of the socket by assigning a local name to an unnamed socket.</p> <p>In the Internet address family, a name consists of several components. For <b>SOCK_DGRAM</b> and <b>SOCK_STREAM</b>, the name consists of three parts: a host address, the protocol number, and a port number which identifies the application. If an application does not care what address is assigned to it, it may specify an Internet address equal to <b>INADDR_ANY</b>, a port equal to 0, or both. If the Internet address is equal to <b>INADDR_ANY</b>, any appropriate network interface will be used; this simplifies application programming in the presence of multi-homed hosts. If the port is specified as 0, the implementation will assign a unique port to the application with a value between 1024 and 30000. The application may use <b>getsockname()</b> after <b>bind()</b> to retrieve the address that has been assigned to it, but note that <b>getsockname()</b> 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.</p>		
Return Value		
<p>If there are no errors, <b>bind()</b> returns 0. Otherwise, it returns -1, and the global variable <b>errno</b> will contain one of the following values.</p>		
Error Codes		
<b>EFAULT</b>	the <b>namelen</b> argument is too small (less than the size of a <b>SOCKADDR</b> ) or the <b>name</b> argument pointer is invalid	
<b>EINVAL</b>	the socket is already bound to an address	
<b>EBADF</b>	the descriptor is not a socket	

<code>closesocket</code>	close a socket	Socket Control																
<p>Syntax</p> <pre>#include &lt;sdksock.h&gt; int closesocket ( int s );</pre>																		
<p>Arguments</p> <p><b>s</b>                                    a descriptor identifying a socket</p>																		
<p>Description</p> <p>This function closes a socket. More precisely, it releases the socket descriptor <code>s</code>, so that further references to <code>s</code> will fail with the error <b>EBADF</b>. If this is the last reference to the underlying socket, the associated naming information and queued data are discarded.</p> <p>The semantics of <code>closesocket()</code> are affected by the socket options <code>SO_LINGER</code> and <code>SO_DONTLINGER</code> as follows:</p> <table border="1"> <thead> <tr> <th>Option</th> <th>Interval</th> <th>Type of close</th> <th>Wait for close?</th> </tr> </thead> <tbody> <tr> <td><code>SO_DONTLINGER</code></td> <td>don't care</td> <td>graceful</td> <td>no</td> </tr> <tr> <td><code>SO_LINGER</code></td> <td>zero</td> <td>hard</td> <td>no</td> </tr> <tr> <td><code>SO_LINGER</code></td> <td>non-zero</td> <td>graceful</td> <td>yes</td> </tr> </tbody> </table> <p>If <code>SO_LINGER</code> is set (i.e., the <code>l_linger</code> field of the <code>linger</code> structure is non-zero) with a zero timeout interval (<code>l_linger</code> is zero), <code>closesocket()</code> 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.</p> <p>If <code>SO_LINGER</code> is set with a non-zero timeout interval, the <code>closesocket()</code> call blocks until the remaining data has been sent or until the timeout expires. This is called a graceful disconnect.</p> <p>If <code>SO_DONTLINGER</code> is set on a stream socket (i.e. the <code>l_onoff</code> field of the <code>linger</code> structure is zero), the <code>closesocket()</code> call will return immediately. However, any data queued for transmission will be sent if possible before the underlying socket is closed. This is also called a graceful disconnect. Note that in this case the implementation may not release the socket and other resources for an arbitrary period, which may affect applications which expect to use all available sockets.</p>			Option	Interval	Type of close	Wait for close?	<code>SO_DONTLINGER</code>	don't care	graceful	no	<code>SO_LINGER</code>	zero	hard	no	<code>SO_LINGER</code>	non-zero	graceful	yes
Option	Interval	Type of close	Wait for close?															
<code>SO_DONTLINGER</code>	don't care	graceful	no															
<code>SO_LINGER</code>	zero	hard	no															
<code>SO_LINGER</code>	non-zero	graceful	yes															
<p>Return Value</p> <p>If there are no errors, <code>closesocket()</code> returns 0. Otherwise, it returns -1, and the global variable <code>errno</code> will contain one of the following values.</p>																		
<p>Error Codes</p> <p><b>EBADF</b>                                    the descriptor is not a socket</p>																		

<b>connect</b>	establish a connection to a peer	<b>Socket Control</b>
Syntax		
<pre>#include &lt;sdksock.h&gt; int connect ( int s, SOCKADDR *name, int namelen );</pre>		
Arguments		
<b>s</b>	a descriptor identifying an unconnected socket	
<b>name</b>	the name of the peer to which the socket is to be connected	
<b>namelen</b>	length of the value in <b>name</b>	
Description		
<p>This function is used to create a connection to the specified foreign association. The parameter <b>s</b> 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, <b>connect()</b> will return the error <b>EADDRNOTAVAIL</b>.</p>		
<p>For stream sockets (type <b>SOCK_STREAM</b>), an active connection is initiated to the foreign host using <b>name</b> (an address in the name space of the socket). When the socket call completes successfully, the socket is ready to send and receive data.</p>		
<p>For a datagram socket (type <b>SOCK_DGRAM</b>), a default destination is set, which will be used on subsequent <b>send()</b> and <b>recv()</b> calls.</p>		
Return Value		
<p>On a blocking socket, the return value indicates success or failure of the connection attempt.</p>		
<p>On a non-blocking socket, if the return value is -1, an application should check <b>errno</b>. If this indicates an error code of <b>EINPROGRESS</b>, then your application can use <b>select()</b> to determine the completion of the connection request by checking if the socket is writeable.</p>		
Error Codes		
<b>EINPROGRESS</b>	(TCP only) socket is nonblocking and a connection attempt would block	
<b>EADDRNOTAVAIL</b>	specified address is not available	
<b>EADDRINUSE</b>	specified address already in use	
<b>ECONNREFUSED</b>	(TCP only) attempt to connect was forcefully rejected by the remote machine	
<b>EISCONN</b>	socket is already connected	
<b>EBADF</b>	descriptor is not a socket	
<b>ETIMEDOUT</b>	(TCP only) attempt to connect timed out without establishing a connection; current timeout value is 30 seconds	

gethostbyname	get host information corresponding to a hostname	Inquiry												
<p>Syntax</p> <pre>#include &lt;sdksock.h&gt; struct hostent *gethostbyname ( char *name );</pre> <p>Arguments</p> <p><b>name</b>                      a pointer to the name of the host</p> <p>Description</p> <p><b>gethostbyname()</b> returns a pointer to the following structure which contains the name(s) and address that correspond to the given address.</p> <pre>struct hostent {     char *    h_name;     char **   h_aliases;     short     h_addrtype;     short     h_length;     char **   h_addr_list; };</pre> <p>The members of this structure are:</p> <table border="1" data-bbox="416 958 1370 1211"> <thead> <tr> <th>Element</th> <th>Usage</th> </tr> </thead> <tbody> <tr> <td><b>h_name</b></td> <td>server name of local system</td> </tr> <tr> <td><b>h_aliases</b></td> <td>a NULL-terminated array of alternate names, currently unused</td> </tr> <tr> <td><b>h_addrtype</b></td> <td>the type of address being returned; this is always <b>AF_INET</b></td> </tr> <tr> <td><b>h_length</b></td> <td>the length (bytes); this is always 4</td> </tr> <tr> <td><b>h_addr_list</b></td> <td>a NULL-terminated list of addresses for the host addresses, returned in network byte order</td> </tr> </tbody> </table> <p>The pointer returned points to a structure that is allocated by the NE-4100-P. Applications must not modify this structure or free any of its components.</p> <p>Return Value</p> <p>If there are no errors, <b>gethostbyname()</b> returns a pointer to the <b>hostent</b> structure described above. Otherwise, it returns a NULL pointer.</p>			Element	Usage	<b>h_name</b>	server name of local system	<b>h_aliases</b>	a NULL-terminated array of alternate names, currently unused	<b>h_addrtype</b>	the type of address being returned; this is always <b>AF_INET</b>	<b>h_length</b>	the length (bytes); this is always 4	<b>h_addr_list</b>	a NULL-terminated list of addresses for the host addresses, returned in network byte order
Element	Usage													
<b>h_name</b>	server name of local system													
<b>h_aliases</b>	a NULL-terminated array of alternate names, currently unused													
<b>h_addrtype</b>	the type of address being returned; this is always <b>AF_INET</b>													
<b>h_length</b>	the length (bytes); this is always 4													
<b>h_addr_list</b>	a NULL-terminated list of addresses for the host addresses, returned in network byte order													



<b>gethostname</b>	return the standard host name for the local machine	<b>Inquiry</b>						
<p>Syntax</p> <pre>#include &lt;sdksock.h&gt; int gethostname ( char *name, int namelen );</pre> <p>Arguments</p> <table> <tr> <td><b>name</b></td> <td>a pointer to a buffer that will receive the host name</td> </tr> <tr> <td><b>namelen</b></td> <td>length of the buffer</td> </tr> </table> <p>Description</p> <p>This routine returns the name of the local host into the buffer specified by the <b>name</b> parameter. The host name is returned as a null-terminated string. The form of the host name is dependent on the socket's implementation—it is a simple host name. However, it is guaranteed that the name returned will be successfully parsed by <b>gethostbyname()</b>.</p> <p>Return Value</p> <p>If there are no errors, <b>gethostname()</b> returns 0. Otherwise, it returns -1, and the global variable <b>errno</b> will contain one of the following values.</p> <p>Error Codes</p> <table> <tr> <td><b>EFAUL</b></td> <td><b>namelen</b> parameter is too small or <b>name</b> pointer is invalid</td> </tr> </table>			<b>name</b>	a pointer to a buffer that will receive the host name	<b>namelen</b>	length of the buffer	<b>EFAUL</b>	<b>namelen</b> parameter is too small or <b>name</b> pointer is invalid
<b>name</b>	a pointer to a buffer that will receive the host name							
<b>namelen</b>	length of the buffer							
<b>EFAUL</b>	<b>namelen</b> parameter is too small or <b>name</b> pointer is invalid							

<b>getpeername</b>	get the address of the peer to which a socket is connected	<b>Inquiry</b>												
<p>Syntax</p> <pre>#include &lt;sdksock.h&gt; int getpeername ( int s, SOCKADDR *name, int *namelen );</pre> <p>Arguments</p> <table> <tr> <td><b>s</b></td> <td>a descriptor identifying a connected socket</td> </tr> <tr> <td><b>name</b></td> <td>the structure which is to receive the name of the peer</td> </tr> <tr> <td><b>namelen</b></td> <td>a pointer to the length of the <b>name</b> structure</td> </tr> </table> <p>Description</p> <p><b>getpeername()</b> retrieves the name of the peer connected to the socket <b>s</b> and stores it in the <b>SOCKADDR</b> identified by <b>name</b>. It is used on a connected datagram or stream socket.</p> <p>On return, the <b>namelen</b> argument contains the actual size of the name returned (bytes).</p> <p>Return Value</p> <p>If there are no errors, <b>getpeername()</b> returns 0. Otherwise, it returns -1, and the global variable <b>errno</b> will contain one of the following values.</p> <p>Error Codes</p> <table> <tr> <td><b>EFAULT</b></td> <td><b>namelen</b> parameter is too small or <b>name</b> pointer is invalid</td> </tr> <tr> <td><b>ENOTCONN</b></td> <td>socket is not connected</td> </tr> <tr> <td><b>EBADF</b></td> <td>descriptor is not a socket</td> </tr> </table>			<b>s</b>	a descriptor identifying a connected socket	<b>name</b>	the structure which is to receive the name of the peer	<b>namelen</b>	a pointer to the length of the <b>name</b> structure	<b>EFAULT</b>	<b>namelen</b> parameter is too small or <b>name</b> pointer is invalid	<b>ENOTCONN</b>	socket is not connected	<b>EBADF</b>	descriptor is not a socket
<b>s</b>	a descriptor identifying a connected socket													
<b>name</b>	the structure which is to receive the name of the peer													
<b>namelen</b>	a pointer to the length of the <b>name</b> structure													
<b>EFAULT</b>	<b>namelen</b> parameter is too small or <b>name</b> pointer is invalid													
<b>ENOTCONN</b>	socket is not connected													
<b>EBADF</b>	descriptor is not a socket													

<b>getsockname</b>	get the local name for a socket	<b>Inquiry</b>
Syntax		
<pre>#include &lt;sdksock.h&gt; int getsockname ( int s, SOCKADDR *name, int *namelen );</pre>		
Arguments		
<b>s</b>	a descriptor identifying a bound socket	
<b>name</b>	receives the address (name) of the socket	
<b>namelen</b>	the length of the <b>name</b> buffer	
Description		
<p><b>getsockname()</b> retrieves the current name for the specified socket descriptor in <b>name</b>. It is used on a bound and/or connected socket specified by the <b>s</b> parameter. The local association is returned. This call is especially useful when a <b>connect()</b> call has been made without first doing a <b>bind()</b>; this call provides the only means by which you can determine the local association which has been set by the system.</p> <p>On return, the <b>namelen</b> argument contains the actual size of the name returned (bytes).</p> <p>If a socket was bound to <b>INADDR_ANY</b>, indicating that any of the host's IP addresses should be used for the socket, <b>getsockname()</b> will not necessarily return information about the host IP address, unless the socket has been connected with <b>connect()</b> or <b>accept()</b>.</p>		
Return Value		
<p>If there are no errors, <b>getsockname()</b> returns 0. Otherwise, it returns -1, and the global variable <b>errno</b> will contain one of the following values.</p>		
Error Codes		
<b>EFAULT</b>	address of <b>name</b> or <b>namelen</b> argument is not large enough	
<b>EBADF</b>	descriptor is not a socket	

<code>getsockopt</code>	retrieve a socket option	Socket Control												
<p>Syntax</p> <pre>#include &lt;sdksock.h&gt; int getsockopt ( int s, int level, int optname, char *optval, int *optlen );</pre>														
<p>Arguments</p> <table> <tr> <td><b>s</b></td> <td>a descriptor identifying a socket</td> </tr> <tr> <td><b>level</b></td> <td>the level at which the option is defined; the only supported levels are <code>SOL_SOCKET</code></td> </tr> <tr> <td><b>optname</b></td> <td>the socket option for which the value is to be retrieved</td> </tr> <tr> <td><b>optval</b></td> <td>a pointer to the buffer in which the value for the requested option is to be returned</td> </tr> <tr> <td><b>optlen</b></td> <td>a pointer to the size of the <code>optval</code> buffer</td> </tr> </table>			<b>s</b>	a descriptor identifying a socket	<b>level</b>	the level at which the option is defined; the only supported levels are <code>SOL_SOCKET</code>	<b>optname</b>	the socket option for which the value is to be retrieved	<b>optval</b>	a pointer to the buffer in which the value for the requested option is to be returned	<b>optlen</b>	a pointer to the size of the <code>optval</code> buffer		
<b>s</b>	a descriptor identifying a socket													
<b>level</b>	the level at which the option is defined; the only supported levels are <code>SOL_SOCKET</code>													
<b>optname</b>	the socket option for which the value is to be retrieved													
<b>optval</b>	a pointer to the buffer in which the value for the requested option is to be returned													
<b>optlen</b>	a pointer to the size of the <code>optval</code> buffer													
<p>Description</p> <p><code>getsockopt()</code> retrieves the current value for a socket option associated with a socket of any type, in any state, and stores the result in <code>optval</code>. Options may exist at multiple protocol levels, but they are always present at the uppermost “socket” level.</p> <p>The value associated with the selected option is returned in the buffer <code>optval</code>. The integer pointed to by <code>optlen</code> should originally contain the size of this buffer; on return, it will be set to the size of the value returned. For <code>SO_LINGER</code>, this will be the size of a struct <code>linger</code>; for all other options it will be the size of an integer.</p> <p>If the option was never set with <code>setsockopt()</code>, then <code>getsockopt()</code> returns the default value for the option.</p> <p>The following socket options are supported for <code>getsockopt()</code>. The Type identifies the type of data addressed by <code>optval</code>:</p> <table border="1"> <thead> <tr> <th>Value</th> <th>Type</th> <th>Meaning</th> </tr> </thead> <tbody> <tr> <td><code>SO_DONTLINGER</code></td> <td>BOOL</td> <td>if true, the <code>SO_LINGER</code> option is disabled</td> </tr> <tr> <td><code>SO_KEEPAALIVE</code></td> <td>BOOL</td> <td>keepalives are being sent</td> </tr> <tr> <td><code>SO_LINGER</code></td> <td>LINGER</td> <td>returns the current linger options</td> </tr> </tbody> </table> <p>Calling <code>getsockopt()</code> with an unsupported option will result in an error code of <code>ENOPROTOOPT</code>.</p>			Value	Type	Meaning	<code>SO_DONTLINGER</code>	BOOL	if true, the <code>SO_LINGER</code> option is disabled	<code>SO_KEEPAALIVE</code>	BOOL	keepalives are being sent	<code>SO_LINGER</code>	LINGER	returns the current linger options
Value	Type	Meaning												
<code>SO_DONTLINGER</code>	BOOL	if true, the <code>SO_LINGER</code> option is disabled												
<code>SO_KEEPAALIVE</code>	BOOL	keepalives are being sent												
<code>SO_LINGER</code>	LINGER	returns the current linger options												
<p>Return Value</p> <p>If there are no errors, <code>getsockopt()</code> returns 0. Otherwise, it returns -1, and the global variable <code>errno</code> will contain one of the following values.</p>														
<p>Error Codes</p> <table> <tr> <td><b>EFAULT</b></td> <td><code>optlen</code> argument was invalid</td> </tr> <tr> <td><b>ENOPROTOOPT</b></td> <td>option is unknown or unsupported</td> </tr> <tr> <td><b>EBADF</b></td> <td>descriptor is not a socket</td> </tr> </table>			<b>EFAULT</b>	<code>optlen</code> argument was invalid	<b>ENOPROTOOPT</b>	option is unknown or unsupported	<b>EBADF</b>	descriptor is not a socket						
<b>EFAULT</b>	<code>optlen</code> argument was invalid													
<b>ENOPROTOOPT</b>	option is unknown or unsupported													
<b>EBADF</b>	descriptor is not a socket													

<b>htonl</b>	convert an unsigned long from host to network byte order	<b>Misc.</b>
<p>Syntax</p> <pre>#include &lt;sdksock.h&gt; u_long htonl ( u_long hostlong );</pre> <p>Arguments</p> <p><b>hostlong</b>                    a 32-bit number in host byte order</p> <p>Description</p> <p>This routine takes a 32-bit number in host byte order and returns a 32-bit number in network byte order.</p> <p>Return Value</p> <p><b>htonl()</b> returns the value in network byte order.</p>		
<b>htons</b>	convert an unsigned short from host to network byte order	<b>Misc.</b>
<p>Syntax</p> <pre>#include &lt;sdksock.h&gt; u_short htons ( u_short hostshort );</pre> <p>Arguments</p> <p><b>hostshort</b>                    a 16-bit number in host byte order</p> <p>Description</p> <p>This routine takes a 16-bit number in host byte order and returns a 16-bit number in network byte order.</p> <p>Return Value</p> <p><b>htons()</b> returns the value in network byte order.</p>		

<code>inet_addr</code>	convert a string containing a dotted address into an long integer	<b>Misc.</b>
<p>Syntax</p> <pre>#include &lt;sdksock.h&gt; unsigned long inet_addr ( char *cp );</pre> <p>Arguments</p> <p><code>cp</code> character string of an Internet address in standard “.” notation</p> <p>Description</p> <p>This function interprets the character string specified by the <code>cp</code> 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).</p> <p>Internet addresses specified using the “.” notation take the form <code>a.b.c.d</code>, such as <code>192.168.127.254</code>.</p> <p>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 <code>d.c.b.a</code>. That is, the bytes on an Intel processor are ordered from right to left.</p> <p>Return Value</p> <p>If there are no errors, <code>inet_addr()</code> 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, <code>inet_addr()</code> returns the value <code>INADDR_ANY</code>.</p>		
<code>inet_ntoa</code>	convert a network address into a string in dotted format	<b>Misc.</b>
<p>Syntax</p> <pre>#include &lt;sdksock.h&gt; char *inet_ntoa ( unsigned long in );</pre> <p>Arguments</p> <p><code>in</code> an Internet host address</p> <p>Description</p> <p>This function takes an Internet address specified by the <code>in</code> parameter. It returns an ASCII string representing the address in “.” notation as <code>a.b.c.d</code>. Note that the string returned by <code>inet_ntoa()</code> resides in memory which is allocated by the sockets implementation. The application should not make any assumptions about the way in which the memory is allocated. The data is guaranteed to be valid until the next socket API call, but no longer.</p> <p>Return Value</p> <p>If there are no errors, <code>inet_ntoa()</code> returns a character pointer to a static buffer containing the text address in standard “.” notation. Otherwise, it returns NULL. The data should be copied before another sockets call is made.</p>		

<code>ioctlsocket</code>	control the mode of a socket	Socket Control						
<p>Syntax</p> <pre>#include &lt;sdksock.h&gt; int ioctlsocket ( int s, long cmd, u_long *argp );</pre>								
<p>Arguments</p> <table> <tr> <td><b>s</b></td> <td>a descriptor identifying a socket</td> </tr> <tr> <td><b>cmd</b></td> <td>the command to perform on the socket <b>s</b></td> </tr> <tr> <td><b>argp</b></td> <td>a pointer to a parameter for <b>cmd</b></td> </tr> </table>			<b>s</b>	a descriptor identifying a socket	<b>cmd</b>	the command to perform on the socket <b>s</b>	<b>argp</b>	a pointer to a parameter for <b>cmd</b>
<b>s</b>	a descriptor identifying a socket							
<b>cmd</b>	the command to perform on the socket <b>s</b>							
<b>argp</b>	a pointer to a parameter for <b>cmd</b>							
<p>Description</p> <p>This routine may be used on any socket in any state. It is used to get or retrieve operating parameters associated with the socket, independent of the protocol and communication subsystem. The following commands are supported:</p> <table border="1"> <thead> <tr> <th>Command</th> <th>Semantics</th> </tr> </thead> <tbody> <tr> <td><b>FIONBIO</b></td> <td>Use this command to enable or disable non-blocking mode on socket <b>s</b>. The parameter <b>argp</b> 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).</td> </tr> </tbody> </table>			Command	Semantics	<b>FIONBIO</b>	Use this command to enable or disable non-blocking mode on socket <b>s</b> . The parameter <b>argp</b> 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).		
Command	Semantics							
<b>FIONBIO</b>	Use this command to enable or disable non-blocking mode on socket <b>s</b> . The parameter <b>argp</b> 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).							
<p>Return Value</p> <p>Upon successful completion, the <code>ioctlsocket()</code> returns 0. Otherwise, it returns -1, and the global variable <code>errno</code> will contain one of the following values.</p>								
<p>Error Codes</p> <table> <tr> <td><b>EINVAL</b></td> <td><b>cmd</b> is not a valid command, <b>argp</b> is not an acceptable parameter for <b>cmd</b>, or the command is not applicable to the type of socket supplied</td> </tr> <tr> <td><b>EBADF</b></td> <td>descriptor <b>s</b> is not a socket</td> </tr> </table>			<b>EINVAL</b>	<b>cmd</b> is not a valid command, <b>argp</b> is not an acceptable parameter for <b>cmd</b> , or the command is not applicable to the type of socket supplied	<b>EBADF</b>	descriptor <b>s</b> is not a socket		
<b>EINVAL</b>	<b>cmd</b> is not a valid command, <b>argp</b> is not an acceptable parameter for <b>cmd</b> , or the command is not applicable to the type of socket supplied							
<b>EBADF</b>	descriptor <b>s</b> is not a socket							

<b>listen</b>	establish a socket to listen for incoming connection	<b>Socket Control</b>
<p>Syntax</p> <pre>#include &lt;sdksock.h&gt; int listen ( int s, int backlog );</pre>		
<p>Arguments</p> <p><b>s</b> descriptor identifying a bound, unconnected socket</p> <p><b>backlog</b> maximum number of connections that can be established from the socket; this is different from the standard BSD socket</p>		
<p>Description</p> <p>To accept connections, a socket is first created with <b>socket()</b>. A backlog for incoming connections is then specified with <b>listen()</b>, and then connections are accepted with <b>accept()</b>. <b>listen()</b> applies only to sockets that support connections (i.e., those of type <b>SOCK_STREAM</b>). The socket <b>s</b> is put into “passive” mode where incoming connections are acknowledged and queued pending acceptance by the process.</p>		
<p>Return Value</p> <p>If there are no errors, <b>listen()</b> returns 0. Otherwise, it returns -1, and the global variable <b>errno</b> will contain one of the following values.</p>		
<p>Error Codes</p> <p><b>EBADF</b> descriptor is not a socket</p> <p><b>EOPNOTSUPP</b> referenced socket is not of a type that supports the <b>listen()</b> operation</p>		

<b>ntohl</b>	convert an unsigned long from network to host byte order	<b>Misc.</b>
<p>Syntax</p> <pre>#include &lt;sdksock.h&gt; u_long ntohl ( u_long netlong );</pre>		
<p>Arguments</p> <p><b>netlong</b> a 32-bit number in network byte order</p>		
<p>Description</p> <p>This routine takes a 32-bit number in network byte order and returns a 32-bit number in host byte order.</p>		
<p>Return Value</p> <p><b>ntohl()</b> returns the value in host byte order.</p>		

<b>ntohs</b>	convert an unsigned short from network to host byte order.	<b>Misc.</b>
<p>Syntax</p> <pre>#include &lt;sdksock.h&gt; u_short ntohs ( u_short netshort );</pre> <p>Arguments</p> <p><b>netshort</b>                    a 16-bit number in network byte order</p> <p>Description</p> <p>This routine takes a 16-bit number in network byte order and returns a 16-bit number in host byte order.</p> <p>Return Value</p> <p><b>ntohs( )</b> returns the value in host byte order.</p>		



<b>recv</b>	receive data from a socket	<b>Data Input/Output</b>														
<p>Syntax</p> <pre>#include &lt;sdksock.h&gt; int recv ( int s, char *buf, int len, int flags );</pre>																
<p>Arguments</p> <table> <tr> <td><b>s</b></td> <td>a descriptor identifying a connected socket</td> </tr> <tr> <td><b>buf</b></td> <td>a buffer for the incoming data</td> </tr> <tr> <td><b>len</b></td> <td>the size of buffer pointed by <b>buf</b></td> </tr> <tr> <td><b>flags</b></td> <td>specifies the way in which the call is made</td> </tr> </table>			<b>s</b>	a descriptor identifying a connected socket	<b>buf</b>	a buffer for the incoming data	<b>len</b>	the size of buffer pointed by <b>buf</b>	<b>flags</b>	specifies the way in which the call is made						
<b>s</b>	a descriptor identifying a connected socket															
<b>buf</b>	a buffer for the incoming data															
<b>len</b>	the size of buffer pointed by <b>buf</b>															
<b>flags</b>	specifies the way in which the call is made															
<p>Description</p> <p>This function is used on datagram or connected stream sockets specified by the <b>s</b> parameter and is used to read incoming data.</p> <p>For sockets of type <b>SOCK_STREAM</b>, all information currently available up to the size of the buffer supplied is returned.</p> <p>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.</p> <p>If no incoming data is available at the socket, the <b>recv()</b> 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 <b>EWOULDBLOCK</b>. The <b>select()</b> calls may be used to determine when more data arrives.</p> <p>If the socket is of type <b>SOCK_STREAM</b> and the remote side has shut down the connection gracefully or the connection has been reset, a <b>recv()</b> will complete immediately with 0 bytes received.</p> <p><b>flags</b> 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 <b>flags</b> parameter. The latter is constructed by “or-ing” any of the following values:</p> <table border="1"> <thead> <tr> <th>Value</th> <th>Meaning</th> </tr> </thead> <tbody> <tr> <td><b>MSG_OOB</b></td> <td>read out-of-band data (<b>SOCK_STREAM</b> only)</td> </tr> </tbody> </table>			Value	Meaning	<b>MSG_OOB</b>	read out-of-band data ( <b>SOCK_STREAM</b> only)										
Value	Meaning															
<b>MSG_OOB</b>	read out-of-band data ( <b>SOCK_STREAM</b> only)															
<p>Return Value</p> <p>If there are no errors, <b>recv()</b> returns the number of bytes received. If the connection has been closed, it returns 0. Otherwise, it returns -1, and the global variable <b>errno</b> will contain one of the following values.</p>																
<p>Error Codes</p> <table> <tr> <td><b>EBADF</b></td> <td>descriptor is not a socket</td> </tr> <tr> <td><b>EFAULT</b></td> <td><b>buf</b> argument pointer is invalid</td> </tr> <tr> <td><b>EOPNOTSUPP</b></td> <td><b>MSG_OOB</b> was specified, but the socket is not of type <b>SOCK_STREAM</b></td> </tr> <tr> <td><b>ESHUTDOWN</b></td> <td>socket has been shutdown; it is not possible to <b>recv()</b> on a socket after <b>shutdown()</b> has been invoked with <b>how</b> set to 0 or 2</td> </tr> <tr> <td><b>EWOULDBLOCK</b></td> <td>socket is marked as non-blocking and the receive operation would block</td> </tr> <tr> <td><b>EIO</b></td> <td><b>MSG_OOB</b> was specified, but has not received out-of-band data</td> </tr> <tr> <td><b>ELENZERO</b></td> <td><b>length</b> argument is zero</td> </tr> </table>			<b>EBADF</b>	descriptor is not a socket	<b>EFAULT</b>	<b>buf</b> argument pointer is invalid	<b>EOPNOTSUPP</b>	<b>MSG_OOB</b> was specified, but the socket is not of type <b>SOCK_STREAM</b>	<b>ESHUTDOWN</b>	socket has been shutdown; it is not possible to <b>recv()</b> on a socket after <b>shutdown()</b> has been invoked with <b>how</b> set to 0 or 2	<b>EWOULDBLOCK</b>	socket is marked as non-blocking and the receive operation would block	<b>EIO</b>	<b>MSG_OOB</b> was specified, but has not received out-of-band data	<b>ELENZERO</b>	<b>length</b> argument is zero
<b>EBADF</b>	descriptor is not a socket															
<b>EFAULT</b>	<b>buf</b> argument pointer is invalid															
<b>EOPNOTSUPP</b>	<b>MSG_OOB</b> was specified, but the socket is not of type <b>SOCK_STREAM</b>															
<b>ESHUTDOWN</b>	socket has been shutdown; it is not possible to <b>recv()</b> on a socket after <b>shutdown()</b> has been invoked with <b>how</b> set to 0 or 2															
<b>EWOULDBLOCK</b>	socket is marked as non-blocking and the receive operation would block															
<b>EIO</b>	<b>MSG_OOB</b> was specified, but has not received out-of-band data															
<b>ELENZERO</b>	<b>length</b> argument is zero															

<b>recvfrom</b>	receive a datagram and store the source address	<b>Data Input/Output</b>				
Syntax						
<pre>#include &lt;sdksock.h&gt; int recvfrom ( int s, char *buf, int len, int flags, SOCKADDR *from, int *fromlen );</pre>						
Arguments						
<b>s</b>	a descriptor identifying a bound socket					
<b>buf</b>	a buffer for the incoming data					
<b>len</b>	the length of <b>buf</b>					
<b>flags</b>	specifies the way in which the call is made					
<b>from</b>	an optional pointer to a buffer which will hold the source address upon return					
<b>fromlen</b>	an optional pointer to the size of the <b>from</b> buffer					
Description						
<p>This function is used to read incoming data on a (possibly connected) socket and capture the address from which the data was sent.</p>						
<p>For sockets of type <b>SOCK_STREAM</b>, all information currently available up to the size of the buffer supplied is returned. The <b>from</b> and <b>fromlen</b> parameters are ignored for <b>SOCK_STREAM</b> sockets.</p>						
<p>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.</p>						
<p>If <b>from</b> is non-zero, and the socket is of type <b>SOCK_DGRAM</b>, the network address of the peer which sent the data is copied to the corresponding <b>SOCKADDR</b>. The value pointed to by <b>fromlen</b> is initialized to the size of this structure, and is modified on return to indicate the actual size of the address stored there.</p>						
<p>If no incoming data is available at the socket, the <b>recvfrom()</b> 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 <b>EWOULDBLOCK</b>. The <b>select()</b> calls may be used to determine when more data arrives.</p>						
<p>If the socket is of type <b>SOCK_STREAM</b> and the remote side has shut down the connection gracefully or the connection has been reset, a <b>recvfrom()</b> will complete immediately with 0 bytes received.</p>						
<p><b>flags</b> 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 <b>flags</b> parameter. The latter is constructed by “or-ing” any of the following values:</p>						
<table border="1"> <thead> <tr> <th data-bbox="416 1624 568 1659">Value</th> <th data-bbox="568 1624 1098 1659">Meaning</th> </tr> </thead> <tbody> <tr> <td data-bbox="416 1659 568 1695">MSG_OOB</td> <td data-bbox="568 1659 1098 1695">read out-of-band data (<b>SOCK_STREAM</b> only)</td> </tr> </tbody> </table>			Value	Meaning	MSG_OOB	read out-of-band data ( <b>SOCK_STREAM</b> only)
Value	Meaning					
MSG_OOB	read out-of-band data ( <b>SOCK_STREAM</b> only)					
Return Value						
<p>If there are no errors, <b>recvfrom()</b> returns the number of bytes received. If the connection has been closed, it returns 0. Otherwise, it returns -1, and the global variable <b>errno</b> will contain one of the following values.</p>						

<b>recvfrom</b>	receive a datagram and store the source address	<b>Data Input/Output</b>														
<p>Error Codes</p> <table> <tr> <td data-bbox="336 421 683 450"><b>EBADF</b></td> <td data-bbox="683 421 1401 450">descriptor is not a socket</td> </tr> <tr> <td data-bbox="336 450 683 479"><b>EFAULT</b></td> <td data-bbox="683 450 1401 479"><b>buf</b> argument pointer is invalid</td> </tr> <tr> <td data-bbox="336 479 683 508"><b>EOPNOTSUPP</b></td> <td data-bbox="683 479 1401 508"><b>MSG_OOB</b> was specified, but socket is not of type <b>SOCK_STREAM</b></td> </tr> <tr> <td data-bbox="336 508 683 575"><b>ESHUTDOWN</b></td> <td data-bbox="683 508 1401 575">socket has been shut down; it is not possible to <b>recvfrom()</b> on a socket after <b>shutdown()</b> has been invoked with <b>how</b> set to 0 or 2</td> </tr> <tr> <td data-bbox="336 575 683 642"><b>EWouldBlock</b></td> <td data-bbox="683 575 1401 642">socket is marked as non-blocking and the receive operation would block</td> </tr> <tr> <td data-bbox="336 642 683 672"><b>EIO</b></td> <td data-bbox="683 642 1401 672"><b>MSG_OOB</b> was specified, but has not received out-of-band data</td> </tr> <tr> <td data-bbox="336 672 683 701"><b>ELENZERO</b></td> <td data-bbox="683 672 1401 701"><b>fromlen</b> argument is zero</td> </tr> </table>			<b>EBADF</b>	descriptor is not a socket	<b>EFAULT</b>	<b>buf</b> argument pointer is invalid	<b>EOPNOTSUPP</b>	<b>MSG_OOB</b> was specified, but socket is not of type <b>SOCK_STREAM</b>	<b>ESHUTDOWN</b>	socket has been shut down; it is not possible to <b>recvfrom()</b> on a socket after <b>shutdown()</b> has been invoked with <b>how</b> set to 0 or 2	<b>EWouldBlock</b>	socket is marked as non-blocking and the receive operation would block	<b>EIO</b>	<b>MSG_OOB</b> was specified, but has not received out-of-band data	<b>ELENZERO</b>	<b>fromlen</b> argument is zero
<b>EBADF</b>	descriptor is not a socket															
<b>EFAULT</b>	<b>buf</b> argument pointer is invalid															
<b>EOPNOTSUPP</b>	<b>MSG_OOB</b> was specified, but socket is not of type <b>SOCK_STREAM</b>															
<b>ESHUTDOWN</b>	socket has been shut down; it is not possible to <b>recvfrom()</b> on a socket after <b>shutdown()</b> has been invoked with <b>how</b> set to 0 or 2															
<b>EWouldBlock</b>	socket is marked as non-blocking and the receive operation would block															
<b>EIO</b>	<b>MSG_OOB</b> was specified, but has not received out-of-band data															
<b>ELENZERO</b>	<b>fromlen</b> argument is zero															

<b>select</b>	determine the status of one or more sockets, waiting if necessary	<b>Data Input/Output</b>																		
<p>Syntax</p> <pre data-bbox="416 405 1361 495">#include &lt;sdksock.h&gt; int select ( int nfd, fd_set *readfd, fd_set *writefd, fd_set *exceptfd, struct timeval *timeout );</pre> <p>Arguments</p> <table border="0" data-bbox="416 546 1399 734"> <tr> <td><b>nfd</b></td> <td>indicates the range of sockets to be checked</td> </tr> <tr> <td><b>readfd</b></td> <td>an optional pointer to a set of sockets to be checked for readability</td> </tr> <tr> <td><b>writefd</b></td> <td>an optional pointer to a set of sockets to be checked for writeability</td> </tr> <tr> <td><b>exceptfd</b></td> <td>an optional pointer to a set of sockets to be checked for errors</td> </tr> <tr> <td><b>timeout</b></td> <td>the maximum time for <b>select()</b> to wait, or NULL for blocking operation</td> </tr> </table> <p>Description</p> <p>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 <b>fd_set</b> structure. Upon return, the structure is updated to reflect the subset of these sockets which meet the specified condition, and <b>select()</b> returns the number of sockets meeting the conditions. A set of macros is provided for manipulating an <b>fd_set</b>. These macros are compatible with those used in the Berkeley software, but the underlying representation is completely different.</p> <p>In each set of sockets, the descriptors from 0 through <b>nfd-1</b> will be examined. This value should not exceed the number of sockets the system allows. For the value of <b>nfd</b>, we recommend that you use the maximum number within the socket sets and add 1. For example, if the maximum number among the socket sets is 7, then 8 should be used as for <b>nfd</b>.</p> <p>Three independent sets of descriptors are watched. Those listed in <b>readfd</b> will be watched to see if characters become available for reading, those in <b>writefd</b> will be watched to see if it is OK to immediately write on them, and those in <b>exceptfd</b> will be watched for exceptions. On exit, the sets are modified in place to indicate which descriptors actually changed status.</p> <p>Any of <b>readfd</b>, <b>writefd</b>, or <b>exceptfd</b> may be given as NULL if no descriptors are of interest.</p> <p>Four macros are defined in the header file <b>sdksock.h</b> for manipulating the descriptor sets. The variable <b>FD_SETSIZE</b> determines the maximum number of descriptors in a set (the default value of <b>FD_SETSIZE</b> is 96). Internally, an <b>fd_set</b> is represented as an array of int's. The macros are as follows:</p> <table border="0" data-bbox="416 1599 1259 1722"> <tr> <td><b>FD_CLR(s, *set)</b></td> <td>removes the descriptor <b>s</b> from set.</td> </tr> <tr> <td><b>FD_ISSET(s, *set)</b></td> <td>nonzero if <b>s</b> is a member of the set, or zero otherwise</td> </tr> <tr> <td><b>FD_SET(s, *set)</b></td> <td>adds descriptor <b>s</b> to set</td> </tr> <tr> <td><b>FD_ZERO(*set)</b></td> <td>initializes the set to the NULL set</td> </tr> </table> <p>The parameter <b>timeout</b> controls how long the <b>select()</b> may take to complete. If <b>timeout</b> is a null pointer, <b>select()</b> will block indefinitely until at least one descriptor meets the specified criteria. Otherwise, <b>timeout</b> points to a struct timeval which specifies the maximum time that <b>select()</b> should wait before returning. If the timeval is initialized to {0, 0}, <b>select()</b> will return immediately; this is used to "poll" the state of the selected sockets.</p>			<b>nfd</b>	indicates the range of sockets to be checked	<b>readfd</b>	an optional pointer to a set of sockets to be checked for readability	<b>writefd</b>	an optional pointer to a set of sockets to be checked for writeability	<b>exceptfd</b>	an optional pointer to a set of sockets to be checked for errors	<b>timeout</b>	the maximum time for <b>select()</b> to wait, or NULL for blocking operation	<b>FD_CLR(s, *set)</b>	removes the descriptor <b>s</b> from set.	<b>FD_ISSET(s, *set)</b>	nonzero if <b>s</b> is a member of the set, or zero otherwise	<b>FD_SET(s, *set)</b>	adds descriptor <b>s</b> to set	<b>FD_ZERO(*set)</b>	initializes the set to the NULL set
<b>nfd</b>	indicates the range of sockets to be checked																			
<b>readfd</b>	an optional pointer to a set of sockets to be checked for readability																			
<b>writefd</b>	an optional pointer to a set of sockets to be checked for writeability																			
<b>exceptfd</b>	an optional pointer to a set of sockets to be checked for errors																			
<b>timeout</b>	the maximum time for <b>select()</b> to wait, or NULL for blocking operation																			
<b>FD_CLR(s, *set)</b>	removes the descriptor <b>s</b> from set.																			
<b>FD_ISSET(s, *set)</b>	nonzero if <b>s</b> is a member of the set, or zero otherwise																			
<b>FD_SET(s, *set)</b>	adds descriptor <b>s</b> to set																			
<b>FD_ZERO(*set)</b>	initializes the set to the NULL set																			

<b>select</b>	determine the status of one or more sockets, waiting if necessary	<b>Data Input/Output</b>				
<p>Return Value</p> <p><b>select</b> ( ) returns the total number of descriptors which are ready and contained in the <b>fd_set</b> structures, 0 if the time limit has expired. Otherwise, it returns -1, and the global variable <b>errno</b> will contain one of the following values.</p> <p>Error Codes</p> <table data-bbox="411 562 1378 622"><tr><td data-bbox="411 562 676 589"><b>EINVAL</b></td><td data-bbox="676 562 1378 589"><b>readfds</b>, <b>wrtefds</b> and <b>exceptfds</b> are all NULL</td></tr><tr><td data-bbox="411 589 676 622"><b>EBADF</b></td><td data-bbox="676 589 1378 622">one of the descriptor sets contains an entry which is not a socket</td></tr></table>			<b>EINVAL</b>	<b>readfds</b> , <b>wrtefds</b> and <b>exceptfds</b> are all NULL	<b>EBADF</b>	one of the descriptor sets contains an entry which is not a socket
<b>EINVAL</b>	<b>readfds</b> , <b>wrtefds</b> and <b>exceptfds</b> are all NULL					
<b>EBADF</b>	one of the descriptor sets contains an entry which is not a socket					

<b>send</b>	send data on a connected socket	<b>Data Input/Output</b>				
Syntax						
<pre>#include &lt;sdksock.h&gt; int send ( int s, const char *buf, int len, int flags );</pre>						
Arguments						
<b>s</b>	a descriptor identifying a connected socket					
<b>buf</b>	a buffer containing the data to be transmitted					
<b>len</b>	the length of the data in <b>buf</b>					
<b>flags</b>	specifies the way in which the call is made					
Description						
<p><b>send()</b> 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.</p>						
<p>Note that the successful completion of a <b>send()</b> does not indicate that the data was successfully delivered.</p>						
<p>If no buffer space is available within the transport system to hold the data to be transmitted, <b>send()</b> will block unless the socket has been placed in a non-blocking I/O mode. On non-blocking <b>SOCK_STREAM</b> 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 <b>select()</b> call may be used to determine when it is possible to send more data.</p>						
<p>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:</p>						
<table border="1"> <thead> <tr> <th data-bbox="416 1189 568 1211">Value</th> <th data-bbox="568 1189 911 1211">Meaning</th> </tr> </thead> <tbody> <tr> <td data-bbox="416 1211 568 1234">MSG_OOB</td> <td data-bbox="568 1211 911 1234">send out-of-band data</td> </tr> </tbody> </table>			Value	Meaning	MSG_OOB	send out-of-band data
Value	Meaning					
MSG_OOB	send out-of-band data					
Return Value						
<p>If there are no errors, <b>send()</b> returns the total number of characters sent (note that this may be less than the number indicated by <b>len</b>). Otherwise, it returns -1, and the global variable <b>errno</b> will contain one of the following values.</p>						
Error Codes						
<b>EFAULT</b>	<b>buf</b> argument is not in a valid part of the user address space					
<b>ENOTCONN</b>	socket is not connected					
<b>EBADF</b>	descriptor is not a socket					
<b>EOPNOTSUPP</b>	<b>MSG_OOB</b> was specified, but the socket is not of type					
	<b>SOCK_STREAM</b>					
<b>ESHUTDOWN</b>	socket has been shut down; it is not possible to <b>send()</b> on a socket					
	after <b>shutdown()</b> has been invoked with <b>how</b> set to 1 or 2					
<b>EWOULDBLOCK</b>	socket is marked as non-blocking and the requested operation					
	would block					
<b>EFBIG</b>	data written exceeds system capacity					

<b>sendto</b>	send data to a specific destination	<b>Data Input/Output</b>												
<p>Syntax</p> <pre>#include &lt;sdksoc.h&gt; int sendto ( int s, char *buf, int len, int flags, SOCKADDR *to, int tolen );</pre>														
<p>Arguments</p> <table border="0"> <tr> <td style="padding-right: 20px;"><b>s</b></td> <td>a descriptor identifying a socket</td> </tr> <tr> <td><b>buf</b></td> <td>a buffer containing the data to be transmitted</td> </tr> <tr> <td><b>len</b></td> <td>length of the data in <b>buf</b>.</td> </tr> <tr> <td><b>flags</b></td> <td>specifies the way in which the call is made</td> </tr> <tr> <td><b>to</b></td> <td>an optional pointer to the address of the target socket</td> </tr> <tr> <td><b>tolen</b></td> <td>the size of the address in <b>to</b></td> </tr> </table>			<b>s</b>	a descriptor identifying a socket	<b>buf</b>	a buffer containing the data to be transmitted	<b>len</b>	length of the data in <b>buf</b> .	<b>flags</b>	specifies the way in which the call is made	<b>to</b>	an optional pointer to the address of the target socket	<b>tolen</b>	the size of the address in <b>to</b>
<b>s</b>	a descriptor identifying a socket													
<b>buf</b>	a buffer containing the data to be transmitted													
<b>len</b>	length of the data in <b>buf</b> .													
<b>flags</b>	specifies the way in which the call is made													
<b>to</b>	an optional pointer to the address of the target socket													
<b>tolen</b>	the size of the address in <b>to</b>													
<p>Description</p> <p><b>sendto()</b> is used on datagram or stream sockets and is used to write outgoing data on a socket. Note that the successful completion of a <b>sendto()</b> does not indicate that the data was successfully delivered.</p> <p><b>sendto()</b> is normally used on a <b>SOCK_DGRAM</b> socket to send a datagram to a specific peer socket identified by the <b>to</b> parameter. On a <b>SOCK_STREAM</b> socket, the <b>to</b> and <b>tolen</b> parameters are ignored; in this case the <b>sendto()</b> is equivalent to <b>send()</b>.</p> <p>To send a broadcast (on a <b>SOCK_DGRAM</b> only), the address in the <b>to</b> parameter should be constructed using the special IP address <b>INADDR_BROADCAST</b> (defined in <b>sdksoc.h</b>) 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.</p> <p>If no buffer space is available within the transport system to hold the data to be transmitted, <b>sendto()</b> will block unless the socket has been placed in a non-blocking I/O mode. On non-blocking <b>SOCK_STREAM</b> 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 <b>select()</b> call may be used to determine when it is possible to send more data.</p> <p><b>flags</b> 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 <b>flags</b> parameter. The latter is constructed by “or-ing” any of the following values:</p> <table border="1" style="margin-left: 20px;"> <thead> <tr> <th>Value</th> <th>Meaning</th> </tr> </thead> <tbody> <tr> <td>MSG_OOB</td> <td>send out-of-band data (<b>SOCK_STREAM</b> only)</td> </tr> </tbody> </table>			Value	Meaning	MSG_OOB	send out-of-band data ( <b>SOCK_STREAM</b> only)								
Value	Meaning													
MSG_OOB	send out-of-band data ( <b>SOCK_STREAM</b> only)													
<p>Return Value</p> <p>If there are no errors, <b>sendto()</b> returns the total number of characters sent (note that this may be less than the number indicated by <b>len</b>). Otherwise, it returns -1, and the global variable <b>errno</b> will contain one of the following values.</p>														

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



setsockopt	set a socket option	Socket Control
Syntax		
<pre>#include &lt;sdksock.h&gt; int setsockopt ( int s, int level, int optname, char *optval, int optlen );</pre>		
Arguments		
<b>s</b>	a descriptor identifying a socket	
<b>level</b>	the level at which the option is defined; the only supported level is <b>SOCKET</b>	
<b>optname</b>	the socket option for which the value is to be set	
<b>optval</b>	a pointer to the buffer in which the value for the requested option is supplied	
<b>optlen</b>	the size of the <b>optval</b> buffer	
Description		
<p><b>setsockopt()</b> sets the current value for a socket option associated with a socket of any type, in any state. Although options may exist at multiple protocol levels, this specification only defines options that exist at the uppermost “socket” level. Options affect socket operations, such as whether keep-connection message is sent in the normal data stream, whether <b>closesocket()</b> operation is graceful, etc.</p>		
<p>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, <b>optval</b> points to a nonzero integer. To disable the option, <b>optval</b> points to an integer equal to zero. <b>optlen</b> should be equal to <b>sizeof(int)</b> for Boolean options. For other options, <b>optval</b> points to a structure that contains the desired value for the option, and <b>optlen</b> is the length of the structure.</p>		
<p><b>SO_LINGER</b> controls the action taken when unsend data is queued on a socket and a <b>closesocket()</b> is performed. See <b>closesocket()</b> for a description of the way in which the <b>SO_LINGER</b> settings affect the semantics of <b>closesocket()</b>. The application sets the desired behavior by creating a struct <b>linger</b> (pointed to by the <b>optval</b> argument) with the following elements:</p>		
<pre>struct linger {     int    l_onoff;     int    l_linger; }</pre>		
<p>To enable <b>SO_LINGER</b>, the application should set <b>l_onoff</b> to a non-zero value, set <b>l_linger</b> to 0 or the desired timeout (seconds), and call <b>setsockopt()</b>. The timeout value should be within 0 and 10 (seconds). To enable <b>SO_DONTLINGER</b> (i.e., disable <b>SO_LINGER</b>) <b>l_onoff</b> should be set to zero and <b>setsockopt()</b> should be called.</p>		
<p>An application may request that the implementation enable the use of “keep-alive” packets on TCP connections by turning on the <b>SO_KEEPALIVE</b> socket option. The precise semantics are implementation-specific, but should conform to section 4.2.3.6 of RFC 1122.</p>		

<b>setsockopt</b>	set a socket option	Socket Control												
<p>The following options are supported for <code>setsockopt()</code>. The Type identifies the type of data addressed by <code>optval</code>.</p> <table border="1"> <thead> <tr> <th>Value</th> <th>Type</th> <th>Meaning</th> </tr> </thead> <tbody> <tr> <td><code>SO_DONTLINGER</code></td> <td>BOOL</td> <td>do not block close waiting for unsent data to be sent; setting this option is equivalent to setting <code>SO_LINGER</code> with <code>l_onoff</code> set to zero</td> </tr> <tr> <td><code>SO_KEEPALIVE</code></td> <td>BOOL</td> <td>send keepalives</td> </tr> <tr> <td><code>SO_LINGER</code></td> <td>LINGER</td> <td>linger on close if unsent data is present</td> </tr> </tbody> </table>			Value	Type	Meaning	<code>SO_DONTLINGER</code>	BOOL	do not block close waiting for unsent data to be sent; setting this option is equivalent to setting <code>SO_LINGER</code> with <code>l_onoff</code> set to zero	<code>SO_KEEPALIVE</code>	BOOL	send keepalives	<code>SO_LINGER</code>	LINGER	linger on close if unsent data is present
Value	Type	Meaning												
<code>SO_DONTLINGER</code>	BOOL	do not block close waiting for unsent data to be sent; setting this option is equivalent to setting <code>SO_LINGER</code> with <code>l_onoff</code> set to zero												
<code>SO_KEEPALIVE</code>	BOOL	send keepalives												
<code>SO_LINGER</code>	LINGER	linger on close if unsent data is present												
<p><b>Return Value</b>            If there are no errors, <code>setsockopt()</code> returns 0. Otherwise, it returns -1, and the global variable <code>errno</code> will contain one of the following values.</p>														
<p><b>Error Codes</b></p> <table> <tbody> <tr> <td><code>EFAULT</code></td> <td><code>optval</code> is not in a valid part of the process address space</td> </tr> <tr> <td><code>EINVAL</code></td> <td><code>level</code> is not valid, or the information in <code>optval</code> is not valid</td> </tr> <tr> <td><code>ENOPROTOOPT</code></td> <td>the option is unknown or unsupported</td> </tr> <tr> <td><code>EBADF</code></td> <td>the descriptor is not a socket</td> </tr> </tbody> </table>			<code>EFAULT</code>	<code>optval</code> is not in a valid part of the process address space	<code>EINVAL</code>	<code>level</code> is not valid, or the information in <code>optval</code> is not valid	<code>ENOPROTOOPT</code>	the option is unknown or unsupported	<code>EBADF</code>	the descriptor is not a socket				
<code>EFAULT</code>	<code>optval</code> is not in a valid part of the process address space													
<code>EINVAL</code>	<code>level</code> is not valid, or the information in <code>optval</code> is not valid													
<code>ENOPROTOOPT</code>	the option is unknown or unsupported													
<code>EBADF</code>	the descriptor is not a socket													

<b>shutdown</b>	disable sends and/or receives on a socket	Socket Control						
<p><b>Syntax</b></p> <pre>#include &lt;sdksock.h&gt; int shutdown ( int s, int how );</pre>								
<p><b>Arguments</b></p> <table> <tbody> <tr> <td><code>s</code></td> <td>a descriptor identifying a socket</td> </tr> <tr> <td><code>how</code></td> <td>a flag that describes what types of operation will no longer be allowed</td> </tr> </tbody> </table>			<code>s</code>	a descriptor identifying a socket	<code>how</code>	a flag that describes what types of operation will no longer be allowed		
<code>s</code>	a descriptor identifying a socket							
<code>how</code>	a flag that describes what types of operation will no longer be allowed							
<p><b>Description</b></p> <p><code>shutdown()</code> is used on all types of sockets to disable reception, transmission, or both.</p> <p>If <code>how</code> 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.</p> <p>If <code>how</code> is 1, subsequent sends are disallowed. For TCP sockets, an FIN will be sent. Setting <code>how</code> to 2 disables both sends and receives as described above.</p> <p>Note that <code>shutdown()</code> does not close the socket, and resources attached to the socket will not be freed until <code>closesocket()</code> is invoked.</p>								
<p><b>Return Value</b>            If there are no errors, <code>shutdown()</code> returns 0. Otherwise, it returns -1, and the global variable <code>errno</code> will contain one of the following values.</p>								
<p><b>Error Codes</b></p> <table> <tbody> <tr> <td><code>EINVAL</code></td> <td><code>how</code> is not valid</td> </tr> <tr> <td><code>ENOTCONN</code></td> <td>socket is not connected (<code>SOCK_STREAM</code> only)</td> </tr> <tr> <td><code>EBADF</code></td> <td>descriptor is not a socket</td> </tr> </tbody> </table>			<code>EINVAL</code>	<code>how</code> is not valid	<code>ENOTCONN</code>	socket is not connected ( <code>SOCK_STREAM</code> only)	<code>EBADF</code>	descriptor is not a socket
<code>EINVAL</code>	<code>how</code> is not valid							
<code>ENOTCONN</code>	socket is not connected ( <code>SOCK_STREAM</code> only)							
<code>EBADF</code>	descriptor is not a socket							

socket	create a socket	Socket Control
Syntax		
<pre>#include &lt;sdksock.h&gt; int socket ( int af, int type, int protocol );</pre>		
Arguments		
<b>af</b>	an address format specification; the only format currently supported is <b>AF_INET</b> , which is the ARPA Internet address format	
<b>type</b>	a type specification for the new socket	
<b>protocol</b>	a particular protocol to be used with the socket, or 0 if the caller does not wish to specify a protocol	
Description		
<p><b>socket ( )</b> 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.</p>		
<p>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.</p>		
<p>The following type specifications are supported:</p>		
<p><b>SOCK_STREAM</b> — provides sequenced, reliable, two-way, connection-based byte streams with an out-of-band data transmission mechanism; uses TCP for the Internet address family</p>		
<p><b>SOCK_DGRAM</b> — supports datagrams, which are connectionless, unreliable buffers of a fixed (typically small) maximum length; uses UDP for the Internet address family</p>		
<p>Sockets of type <b>SOCK_STREAM</b> 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 <b>connect ( )</b> call. Once connected, data may be transferred using <b>send ( )</b> and <b>recv ( )</b> calls. When a session has been completed, a <b>closesocket ( )</b> must be performed. These communication protocols have been developed for <b>SOCK_STREAM</b> to ensure that data is not lost or duplicated.</p>		
<p><b>SOCK_DGRAM</b> sockets allow sending and receiving of datagrams to and from arbitrary peers using <b>sendto ( )</b> and <b>recvfrom ( )</b>. If a connection is established between the socket and a specific peer using <b>connect ( )</b>, datagrams may be sent to that peer using <b>send ( )</b> and may be received from (only) this peer using <b>recv ( )</b>.</p>		
Return Value		
<p>If there are no errors, <b>socket ( )</b> returns a descriptor referencing the new socket. Otherwise, it returns -1, and the global variable <b>errno</b> will contain one of the following values.</p>		
Error Codes		
<b>EMFILE</b>	no more file descriptors are available	
<b>EPROTONOSUPPORT</b>	the specified address family or protocol is not supported	

## Simplified Socket Library Reference

<b>net_get_gateway</b>	get local default gateway	<b>Port Inquiry</b>
<p>Syntax</p> <pre>#include &lt;sdksys.h&gt; u_long net_get_gateway ( void );</pre> <p>Arguments</p> <p>N/A</p> <p>Description</p> <p>get local default gateway</p> <p>Return Value</p> <p>default gateway IP address</p>		
<b>net_get_IP</b>	get local IP address	<b>Port Inquiry</b>
<p>Syntax</p> <pre>#include &lt;sdksys.h&gt; u_long net_get_IP ( void );</pre> <p>Arguments</p> <p>N/A</p> <p>Description</p> <p>get local IP address</p> <p>Return Value</p> <p>local IP address</p>		
<b>net_get_MAC_address</b>	get MAC address	<b>Port Inquiry</b>
<p>Syntax</p> <pre>#include &lt;sdksys.h&gt; void net_get_MAC_address ( u_char *mac );</pre> <p>Arguments</p> <p><b>mac</b> get MAC address buffer pointer; this buffer must be 6-byte length</p> <p>Description</p> <p>get MAC address</p> <p>Return Value</p> <p>system copies the host MAC address to the mac input buffer</p>		

<b>net_get_netmask</b>	get local subnet mask	<b>Port Inquiry</b>
<p>Syntax</p> <pre>#include &lt;sdksys.h&gt; u_long net_get_netmask ( void );</pre> <p>Arguments</p> <p>N/A</p> <p>Description</p> <p>get local subnet mask</p> <p>Return Value</p> <p>local netmask</p>		
<b>tcp_close</b>	close a local TCP port	<b>Socket Control</b>
<p>Syntax</p> <pre>#include &lt;sdknet.h&gt; int tcp_close ( int handle );</pre> <p>Arguments</p> <p><b>handle</b>                    the value returned from <b>tcp_open()</b></p> <p>Description</p> <p>close a local TCP port</p> <p>Return Value</p> <p>0                            OK</p> <p>-1                           error handle number</p>		
<b>tcp_connect</b>	connect to specific host IP and port	<b>Socket Control</b>
<p>Syntax</p> <pre>#include &lt;sdknet.h&gt; int tcp_connect ( int handle, u_long rip, int rport, long tout );</pre> <p>Arguments</p> <p><b>handle</b>                    the value return from <b>tcp_open()</b></p> <p><b>rip</b>                        remote host IP address that user wants to link</p> <p><b>rport</b>                     remote host TCP port number</p> <p><b>tout</b>                      wait for TCP connection time out value (milliseconds); 0 will wait for OK or fail</p> <p>Description</p> <p>connect to specific host IP and port</p> <p>Return Value</p> <p>1                            connect OK</p> <p>0                            connect fail</p> <p>-1                           error handle number</p> <p>-2                           this handle is not a TCP handle</p> <p>-3                           timeout counter reached</p> <p>-4                           error state; already connected</p> <p>-5                           the <b>rip:rport</b> already in use</p>		

<code>tcp_connect_nowait</code>	connect to specific host IP and port no wait	Socket Control										
<p>Syntax</p> <pre>#include &lt;sdknet.h&gt; int tcp_connect_nowait ( int handle, u_long rip, int rport );</pre>												
<p>Arguments</p> <table> <tr> <td><code>handle</code></td> <td>the value returned from <code>tcp_open()</code></td> </tr> <tr> <td><code>rip</code></td> <td>remote host IP address that user wants to link to</td> </tr> <tr> <td><code>rport</code></td> <td>remote host's TCP port number</td> </tr> </table>			<code>handle</code>	the value returned from <code>tcp_open()</code>	<code>rip</code>	remote host IP address that user wants to link to	<code>rport</code>	remote host's TCP port number				
<code>handle</code>	the value returned from <code>tcp_open()</code>											
<code>rip</code>	remote host IP address that user wants to link to											
<code>rport</code>	remote host's TCP port number											
<p>Description</p> <p>connect to specific host's IP and port without waiting</p>												
<p>Return Value</p> <table> <tr> <td>0</td> <td>start to connect</td> </tr> <tr> <td>-1</td> <td>error handle number</td> </tr> <tr> <td>-2</td> <td>error argument</td> </tr> <tr> <td>-3</td> <td>error state; already connected</td> </tr> <tr> <td>-4</td> <td>the <code>rip:rport</code> is already in use</td> </tr> </table>			0	start to connect	-1	error handle number	-2	error argument	-3	error state; already connected	-4	the <code>rip:rport</code> is already in use
0	start to connect											
-1	error handle number											
-2	error argument											
-3	error state; already connected											
-4	the <code>rip:rport</code> is already in use											
<code>tcp_get_remote</code>	get connected host's IP and port	Socket Inquiry										
<p>Syntax</p> <pre>#include &lt;sdknet.h&gt; int tcp_get_remote ( int handle, u_long *rip, int *rport )</pre>												
<p>Arguments</p> <table> <tr> <td><code>handle</code></td> <td>the value returned from <code>tcp_open()</code></td> </tr> <tr> <td><code>rip</code></td> <td>connected host's IP address pointer</td> </tr> <tr> <td><code>rport</code></td> <td>connected host's TCP port number pointer</td> </tr> </table>			<code>handle</code>	the value returned from <code>tcp_open()</code>	<code>rip</code>	connected host's IP address pointer	<code>rport</code>	connected host's TCP port number pointer				
<code>handle</code>	the value returned from <code>tcp_open()</code>											
<code>rip</code>	connected host's IP address pointer											
<code>rport</code>	connected host's TCP port number pointer											
<p>Description</p> <p>get connected host's IP and port</p>												
<p>Return Value</p> <table> <tr> <td>0</td> <td>get ok</td> </tr> <tr> <td>-1</td> <td>error handle</td> </tr> <tr> <td>-2</td> <td>error argument</td> </tr> <tr> <td>-3</td> <td>no connection</td> </tr> </table>			0	get ok	-1	error handle	-2	error argument	-3	no connection		
0	get ok											
-1	error handle											
-2	error argument											
-3	no connection											

<b>tcp_iqueue</b>	get the length of data accumulated in TCP driver's input buffer	<b>Socket Inquiry</b>
<p>Syntax</p> <pre>#include &lt;sdknet.h&gt; int tcp_iqueue ( int handle );</pre> <p>Arguments</p> <p><b>handle</b>                      the value returned from <b>tcp_open()</b></p> <p>Description</p> <p>get the length of data accumulated in TCP driver's input buffer</p> <p>Return Value</p> <p><b>&gt;=0</b>                      TCP input buffer queued data size</p> <p><b>-1</b>                          error handle number</p> <p><b>-2</b>                          this is not a TCP handle</p> <p><b>-3</b>                          TCP not connected</p>		
<b>tcp_listen</b>	place a socket in a state where it is listening for an incoming connection	<b>Socket Control</b>
<p>Syntax</p> <pre>#include &lt;sdknet.h&gt; int tcp_listen ( int handle, long tout );</pre> <p>Arguments</p> <p><b>handle</b>                      the return value from <b>tcp_open()</b></p> <p><b>tout</b>                          wait for listen time out value (milliseconds); 0 will wait for someone to connect</p> <p>Description</p> <p>places a socket a state where it is listening for an incoming connection</p> <p>Return Value</p> <p><b>1</b>                              connect OK or already connected</p> <p><b>0</b>                              connect fail</p> <p><b>-1</b>                              error handle number</p> <p><b>-2</b>                              this handle is not a TCP handle</p> <p><b>-3</b>                              timeout counter reached</p> <p><b>-4</b>                              error state; already connected</p>		

<b>tcp_listen_nowait</b>	place a socket in a state where it is listening for an incoming connection without waiting	<b>Socket Control</b>
<p>Syntax</p> <pre>#include &lt;sdknet.h&gt; int tcp_listen_nowait ( int handle );</pre> <p>Arguments</p> <p><b>handle</b>                      the value returned from <code>tcp_open()</code></p> <p>Description</p> <p>place a socket a state where it is listening for an incoming connection without waiting</p> <p>Return Value</p> <p>0                              start to listen  -1                              error handle number  -2                              this handle is not a TCP handle  -3                              error state; already connected</p>		

<b>tcp_listento</b>	listen for a specific incoming connection	<b>Socket Control</b>
<p>Syntax</p> <pre>#include &lt;sdknet.h&gt; int tcp_listento ( int handle, u_long rip, int rport, long tout );</pre> <p>Arguments</p> <p><b>handle</b>                      the value returned from <code>tcp_open()</code>  <b>rip</b>                              remote host IP address that user wants to link to; 0 indicates any remote IP address  <b>rport</b>                          remote host TCP port number; 0 indicates any TCP port number  <b>tout</b>                              wait for listen timeout value (milliseconds)</p> <p>Description</p> <p>listen for a specific incoming connection</p> <p>Return Value</p> <p>1                              connect OK or already connected  0                              connect fail  -1                              error handle number  -2                              this handle is not a TCP handle  -3                              timeout counter reached  -4                              error state; already connected</p>		



<b>tcp_listento_nowait</b>	listen for a specific incoming connection without waiting	<b>Socket Control</b>														
<p>Syntax</p> <pre>#include &lt;sdknet.h&gt; int tcp_listento_nowait ( int handle, u_long rip, int rport );</pre> <p>Arguments</p> <table> <tr> <td><b>handle</b></td> <td>the value returned from <code>tcp_open()</code></td> </tr> <tr> <td><b>rip</b></td> <td>remote host IP address that user wants to link to; 0 indicates any remote IP address</td> </tr> <tr> <td><b>rport</b></td> <td>remote host's TCP port number; 0 indicates any TCP port number</td> </tr> </table> <p>Description</p> <p>listen for a specific incoming connection no wait</p> <p>Return Value</p> <table> <tr> <td>0</td> <td>start to listen</td> </tr> <tr> <td>-1</td> <td>error handle number</td> </tr> <tr> <td>-2</td> <td>this handle is not a TCP handle</td> </tr> <tr> <td>-3</td> <td>error state; already connected</td> </tr> </table>			<b>handle</b>	the value returned from <code>tcp_open()</code>	<b>rip</b>	remote host IP address that user wants to link to; 0 indicates any remote IP address	<b>rport</b>	remote host's TCP port number; 0 indicates any TCP port number	0	start to listen	-1	error handle number	-2	this handle is not a TCP handle	-3	error state; already connected
<b>handle</b>	the value returned from <code>tcp_open()</code>															
<b>rip</b>	remote host IP address that user wants to link to; 0 indicates any remote IP address															
<b>rport</b>	remote host's TCP port number; 0 indicates any TCP port number															
0	start to listen															
-1	error handle number															
-2	this handle is not a TCP handle															
-3	error state; already connected															
<b>tcp_ofree</b>	get amount of free space in TCP driver's input buffer	<b>Socket Inquiry</b>														
<p>Syntax</p> <pre>#include &lt;sdknet.h&gt; int tcp_ofree ( int handle );</pre> <p>Arguments</p> <table> <tr> <td><b>handle</b></td> <td>the value returned from <code>tcp_open()</code></td> </tr> </table> <p>Description</p> <p>get amount of free space in TCP driver's input buffer</p> <p>Return Value</p> <table> <tr> <td>&gt;=0</td> <td>TCP output buffer's free size</td> </tr> <tr> <td>-1</td> <td>error handle number</td> </tr> <tr> <td>-2</td> <td>this is not a TCP handle</td> </tr> <tr> <td>-3</td> <td>TCP not connected</td> </tr> </table>			<b>handle</b>	the value returned from <code>tcp_open()</code>	>=0	TCP output buffer's free size	-1	error handle number	-2	this is not a TCP handle	-3	TCP not connected				
<b>handle</b>	the value returned from <code>tcp_open()</code>															
>=0	TCP output buffer's free size															
-1	error handle number															
-2	this is not a TCP handle															
-3	TCP not connected															
<b>tcp_open</b>	open a local TCP port	<b>Socket Control</b>														
<p>Syntax</p> <pre>#include &lt;sdknet.h&gt; int tcp_open ( int port );</pre> <p>Arguments</p> <table> <tr> <td><b>port</b></td> <td>local TCP port number</td> </tr> </table> <p>Description</p> <p>open a local TCP port</p> <p>Return Value</p> <table> <tr> <td>&gt;=0</td> <td>open handle</td> </tr> <tr> <td>-1</td> <td>open fail</td> </tr> </table>			<b>port</b>	local TCP port number	>=0	open handle	-1	open fail								
<b>port</b>	local TCP port number															
>=0	open handle															
-1	open fail															

<b>tcp_recv</b>	receive data from a connected socket	<b>Data Input/Output</b>														
<p>Syntax</p> <pre>#include &lt;sdknet.h&gt; int tcp_recv ( int handle, char *buffer, int len );</pre> <p>Arguments</p> <table border="0"> <tr> <td><b>handle</b></td> <td>the value returned from <b>tcp_open()</b></td> </tr> <tr> <td><b>buffer</b></td> <td>the pointer to buffer for incoming data</td> </tr> <tr> <td><b>len</b></td> <td>the size of <b>buffer</b> (bytes)</td> </tr> </table> <p>Description</p> <p>receives data from a connected socket</p> <p>Return Value</p> <table border="0"> <tr> <td><b>&gt;=0</b></td> <td>received data length</td> </tr> <tr> <td><b>-1</b></td> <td>error handle number</td> </tr> <tr> <td><b>-2</b></td> <td>error argument</td> </tr> <tr> <td><b>-3</b></td> <td>TCP not connected</td> </tr> </table>			<b>handle</b>	the value returned from <b>tcp_open()</b>	<b>buffer</b>	the pointer to buffer for incoming data	<b>len</b>	the size of <b>buffer</b> (bytes)	<b>&gt;=0</b>	received data length	<b>-1</b>	error handle number	<b>-2</b>	error argument	<b>-3</b>	TCP not connected
<b>handle</b>	the value returned from <b>tcp_open()</b>															
<b>buffer</b>	the pointer to buffer for incoming data															
<b>len</b>	the size of <b>buffer</b> (bytes)															
<b>&gt;=0</b>	received data length															
<b>-1</b>	error handle number															
<b>-2</b>	error argument															
<b>-3</b>	TCP not connected															
<b>tcp_send</b>	sends data to a connected socket	<b>Data Input/Output</b>														
<p>Syntax</p> <pre>#include &lt;sdknet.h&gt; int tcp_send ( int handle, char *buffer, int len );</pre> <p>Arguments</p> <table border="0"> <tr> <td><b>handle</b></td> <td>the value return from <b>tcp_open()</b></td> </tr> <tr> <td><b>buffer</b></td> <td>the pointer to buffer for outgoing data</td> </tr> <tr> <td><b>len</b></td> <td>the length of data in buffer to be sent (bytes)</td> </tr> </table> <p>Description</p> <p>sends data on a connected socket</p> <p>Return Value</p> <table border="0"> <tr> <td><b>&gt;=0</b></td> <td>outgoing data length</td> </tr> <tr> <td><b>-1</b></td> <td>error handle number</td> </tr> <tr> <td><b>-2</b></td> <td>error argument</td> </tr> <tr> <td><b>-3</b></td> <td>TCP not connected</td> </tr> </table>			<b>handle</b>	the value return from <b>tcp_open()</b>	<b>buffer</b>	the pointer to buffer for outgoing data	<b>len</b>	the length of data in buffer to be sent (bytes)	<b>&gt;=0</b>	outgoing data length	<b>-1</b>	error handle number	<b>-2</b>	error argument	<b>-3</b>	TCP not connected
<b>handle</b>	the value return from <b>tcp_open()</b>															
<b>buffer</b>	the pointer to buffer for outgoing data															
<b>len</b>	the length of data in buffer to be sent (bytes)															
<b>&gt;=0</b>	outgoing data length															
<b>-1</b>	error handle number															
<b>-2</b>	error argument															
<b>-3</b>	TCP not connected															

<b>tcp_state</b>	get TCP state	<b>Socket Inquiry</b>																
<p>Syntax</p> <pre>#include &lt;sdknet.h&gt; int tcp_state ( int handle );</pre> <p>Arguments</p> <p><b>handle</b>                    the value returned from <b>tcp_open()</b></p> <p>Description</p> <p>get TCP state</p> <p>Return Value</p> <table> <tr><td>0</td><td>TCP closed</td></tr> <tr><td>1</td><td>TCP listen</td></tr> <tr><td>2</td><td>TCP connecting</td></tr> <tr><td>3</td><td>TCP connected</td></tr> <tr><td>4</td><td>TCP close wait (remote closed)</td></tr> <tr><td>5</td><td>TCP closing</td></tr> <tr><td>-1</td><td>error handle</td></tr> <tr><td>-2</td><td>this handle is not a TCP handle</td></tr> </table>			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
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																	
<b>udp_close</b>	close a local UDP port	<b>Socket Control</b>																
<p>Syntax</p> <pre>#include &lt;sdknet.h&gt; int udp_close ( int handle );</pre> <p>Arguments</p> <p><b>handle</b>                    the value return from <b>udp_open()</b></p> <p>Description</p> <p>close a local UDP port</p> <p>Return Value</p> <table> <tr><td>0</td><td>close OK</td></tr> <tr><td>-1</td><td>error handle number</td></tr> </table>			0	close OK	-1	error handle number												
0	close OK																	
-1	error handle number																	
<b>udp_iqueue</b>	get length of data accumulated in UDP driver's input buffer	<b>Socket Inquiry</b>																
<p>Syntax</p> <pre>#include &lt;sdknet.h&gt; int udp_iqueue ( int handle )</pre> <p>Arguments</p> <p><b>handle</b>                    the value returned from <b>udp_open()</b></p> <p>Description</p> <p>get the length of data accumulated in UDP driver's input buffer</p> <p>Return Value</p> <table> <tr><td>&gt;=0</td><td>UDP input buffer queued data size</td></tr> <tr><td>-1</td><td>error handle number</td></tr> <tr><td>-2</td><td>this is not a UDP handle</td></tr> </table>			>=0	UDP input buffer queued data size	-1	error handle number	-2	this is not a UDP handle										
>=0	UDP input buffer queued data size																	
-1	error handle number																	
-2	this is not a UDP handle																	

<b>udp_ofree</b>	get amount of free space in UDP driver's input buffer	<b>Socket Inquiry</b>
<p>Syntax</p> <pre>#include &lt;sdknet.h&gt; int udp_ofree ( int handle );</pre> <p>Arguments</p> <p><b>handle</b>                    the value returned from <b>udp_open()</b></p> <p>Description</p> <p>amount of free space in UDP driver's input buffer</p> <p>Return Value</p> <p><b>&gt;=0</b>                    UDP output buffer free size  <b>-1</b>                        error handle number  <b>-2</b>                        this is not a UDP handle</p>		
<b>udp_open</b>	open a local UDP port	<b>Socket Inquiry</b>
<p>Syntax</p> <pre>#include &lt;sdknet.h&gt; int udp_open ( int port );</pre> <p>Arguments</p> <p><b>port</b>                      local UDP port number</p> <p>Description</p> <p>open a local UDP port</p> <p>Return Value</p> <p><b>&gt;=0</b>                      open handle  <b>-1</b>                         open fail</p>		
<b>udp_rcv</b>	receive data from a specific source address	<b>Data Input/Output</b>
<p>Syntax</p> <pre>#include &lt;sdknet.h&gt; int udp_rcv ( int handle, u_long *rip, int *rport, char *buf, int len );</pre> <p>Arguments</p> <p><b>handle</b>                    the value returned from <b>udp_open()</b>  <b>rip</b>                        the remote host's IP address  <b>rport</b>                    pointer to the remote UDP port number  <b>buf</b>                        pointer to buffer for incoming data  <b>len</b>                        the length of <b>buf</b> (bytes)</p> <p>Description</p> <p>receive data from a specific source address</p> <p>Return Value</p> <p><b>&gt;= 0</b>                    length of data received  <b>-1</b>                        receive failed</p>		

<code>udp_send</code>	sends data to a specific destination	Data Input/Output										
Syntax <pre>#include &lt;sdknet.h&gt; int udp_send ( int handle, u_long rip, int rport, char *buf, int len );</pre>												
Arguments <table> <tr> <td><code>handle</code></td> <td>the value returned from <code>udp_open()</code></td> </tr> <tr> <td><code>rip</code></td> <td>destination host IP address</td> </tr> <tr> <td><code>rport</code></td> <td>destination host UDP port number</td> </tr> <tr> <td><code>buf</code></td> <td>pointer to buffer for outgoing data</td> </tr> <tr> <td><code>len</code></td> <td>send data length (bytes)</td> </tr> </table>			<code>handle</code>	the value returned from <code>udp_open()</code>	<code>rip</code>	destination host IP address	<code>rport</code>	destination host UDP port number	<code>buf</code>	pointer to buffer for outgoing data	<code>len</code>	send data length (bytes)
<code>handle</code>	the value returned from <code>udp_open()</code>											
<code>rip</code>	destination host IP address											
<code>rport</code>	destination host UDP port number											
<code>buf</code>	pointer to buffer for outgoing data											
<code>len</code>	send data length (bytes)											
Description send data to a specific destination												
Return Value <table> <tr> <td><code>&gt;= 0</code></td> <td>length of data sent out</td> </tr> <tr> <td><code>-1</code></td> <td>send failed</td> </tr> </table>			<code>&gt;= 0</code>	length of data sent out	<code>-1</code>	send failed						
<code>&gt;= 0</code>	length of data sent out											
<code>-1</code>	send failed											

## System Control Library Reference

<code>sys_calloc</code>	allocates an array in memory with elements initialized to 0					
Syntax <pre>#include &lt;sdksys.h&gt; void *sys_calloc ( unsigned nelem, unsigned elsize );</pre>						
Arguments <table> <tr> <td><code>nelem</code></td> <td>number of elements to be allocated</td> </tr> <tr> <td><code>elsize</code></td> <td>length of each element (bytes)</td> </tr> </table>			<code>nelem</code>	number of elements to be allocated	<code>elsize</code>	length of each element (bytes)
<code>nelem</code>	number of elements to be allocated					
<code>elsize</code>	length of each element (bytes)					
Description The <code>sys_calloc()</code> function allocates space for <code>nelem</code> objects, each <code>elsize</code> bytes in length. The result is identical to calling <code>sys_malloc()</code> with an argument of ' <code>nelem * elsize</code> ', with the exception that the allocated memory is explicitly initialized to zero bytes.						
Return Value The <code>sys_calloc()</code> function returns a pointer to the allocated memory if successful; otherwise a NULL pointer is returned.						

<code>sys_clock_ms</code>	read the time count (milliseconds) from power-up	
Syntax <pre>#include &lt;sdksys.h&gt; unsigned long sys_clock_ms ( void );</pre>		
Arguments N/A		
Description read the NE-4100-P's time count (milliseconds) from power-up		
Return Value This function returns the time counter in milliseconds.		

<b>sys_clock_s</b>	read the time count (seconds) from power-up	
<p>Syntax</p> <pre>#include &lt;sdksys.h&gt; unsigned long sys_clock_s ( void );</pre> <p>Arguments</p> <p>N/A</p> <p>Description</p> <p>read the NE-4100-P's time count (seconds) from power-up</p> <p>Return Value</p> <p>This function returns the time counter in seconds.</p>		
<b>sys_exit</b>	exit application	
<p>Syntax</p> <pre>#include &lt;sdksys.h&gt; void sys_exit ( void );</pre> <p>Arguments</p> <p>N/A</p> <p>Description</p> <p>exit user application and return to kernel; it will stop the user application</p> <p>Return Value</p> <p>N/A</p>		
<b>sys_free</b>	deallocate or free a memory block	
<p>Syntax</p> <pre>#include &lt;sdksys.h&gt; void sys_free ( void *ptr );</pre> <p>Arguments</p> <p><b>ptr</b> pointer to a memory returned by <b>sys_malloc()</b> or <b>sys_calloc()</b></p> <p>Description</p> <p>The <b>sys_free()</b> function causes the allocated memory referenced by <b>ptr</b> to be made available for future allocations.</p> <p>Return Value</p> <p>N/A</p>		

<b>sys_get_info</b>	get general server information	
<p>Syntax</p> <pre data-bbox="416 371 1098 427">#include &lt;sdksys.h&gt; int sys_get_info ( struct sdk_sysinfo *info );</pre> <p>Arguments</p> <p><b>info</b> a pointer to a buffer to receive the general server information</p> <p>Description</p> <p>General server information for the NE-4100-P is returned with the following structure:</p> <pre data-bbox="416 611 1401 1055">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 */ };</pre> <p>The version information is stored with the following structure:</p> <pre data-bbox="416 1122 932 1272">struct sdk_version {     unsigned short   ext_version;     unsigned char    sub_version;     unsigned char    main_version; };</pre> <p>For version 1.20.3, the <code>main_version</code> is 1, <code>sub_version</code> is 20 and <code>ext_version</code> is 3.</p> <p>Return Value</p> <p><code>sys_get_info()</code> returns the length of the information structure. A return value of 0 indicates the argument is invalid.</p>		

<b>sys_get_LastErrno</b>	get last error number related to a socket					
<p>Syntax</p> <pre>#include &lt;sdksys.h&gt; int sys_get_LastErrno ( int socket, int *err );</pre>						
<p>Arguments</p> <table> <tr> <td><b>socket</b></td> <td>a descriptor identifying a socket</td> </tr> <tr> <td><b>err</b></td> <td>a pointer to where to place the last error number</td> </tr> </table>			<b>socket</b>	a descriptor identifying a socket	<b>err</b>	a pointer to where to place the last error number
<b>socket</b>	a descriptor identifying a socket					
<b>err</b>	a pointer to where to place the last error number					
<p>Description</p> <p>Most sockets APIs will put the error reason as the error number in the global variable <b>errno</b>. In some multi-threading application, the variable may be overwritten suddenly by another thread. In such case, you can use this function to retrieve the last error encountered by the specified socket. On return, the error number is placed in the space specified by <b>err</b>.</p> <p>Note that the returned error number is meaningful only when the last socket operation on specified socket failed. Otherwise, the returned error number is undefined.</p>						
<p>Return Value</p> <table> <tr> <td>-3</td> <td>the pointer is invalid</td> </tr> <tr> <td>0</td> <td>OK</td> </tr> </table>			-3	the pointer is invalid	0	OK
-3	the pointer is invalid					
0	OK					

<b>sys_get_SerialType</b>	get async port interface signal type											
<p>Syntax</p> <pre>#include &lt;sdksys.h&gt; int sys_get_SerialType ( int port );</pre>												
<p>Arguments</p> <table> <tr> <td><b>port</b></td> <td>async serial port number</td> </tr> </table>			<b>port</b>	async serial port number								
<b>port</b>	async serial port number											
<p>Description</p> <p>get async port interface signal type</p>												
<p>Return Value</p> <table> <tr> <td>0</td> <td>RS-23</td> </tr> <tr> <td>1</td> <td>RS-42</td> </tr> <tr> <td>2</td> <td>RS-485 2-wire</td> </tr> <tr> <td>3</td> <td>RS-485 4-wire</td> </tr> <tr> <td>-1</td> <td>bad port</td> </tr> </table>			0	RS-23	1	RS-42	2	RS-485 2-wire	3	RS-485 4-wire	-1	bad port
0	RS-23											
1	RS-42											
2	RS-485 2-wire											
3	RS-485 4-wire											
-1	bad port											



<b>sys_getFreeMemSize</b>	get available memory status used by <b>sys_malloc()</b> and <b>sys_calloc()</b>							
<p>Syntax</p> <pre>#include &lt;sdksys.h&gt; int sys_getFreeMemSize ( long *total_size, long *max_block_size );</pre> <p>Arguments</p> <table> <tr> <td><b>total_size</b></td> <td>pointer to buffer to retrieve total size of free memory (bytes)</td> </tr> <tr> <td><b>max_block_size</b></td> <td>pointer to buffer to retrieve the maximum size of free memory blocks</td> </tr> </table> <p>Description</p> <p>When a program requests to allocate memory with <b>sys_malloc()</b> or so on, the system will return a block of continuous free memory. After allocation and deallocation occurs several times, the entire memory space may contain several small blocks of free memory of different sizes. Although the total free memory may still be large enough, <b>sys_malloc()</b> may fail if there is no contiguous block large enough for the desired size.</p> <p>After <b>sys_getFreeMemorySize()</b> is called, the total size of free memory is put in the buffer pointed by <b>total_size</b>, and the maximum size of free memory blocks is put in the buffer pointed by <b>max_block_size</b>.</p> <p>This function is useful for debugging and may be used to determine if there is a memory leak.</p> <p>Return Value</p> <table> <tr> <td>0</td> <td>This function always returns 0.</td> </tr> </table>			<b>total_size</b>	pointer to buffer to retrieve total size of free memory (bytes)	<b>max_block_size</b>	pointer to buffer to retrieve the maximum size of free memory blocks	0	This function always returns 0.
<b>total_size</b>	pointer to buffer to retrieve total size of free memory (bytes)							
<b>max_block_size</b>	pointer to buffer to retrieve the maximum size of free memory blocks							
0	This function always returns 0.							

<b>sys_GetServersIp</b>	retrieve the DNS server's and time server's address supplied by DHCP/BOOTP server									
<p>Syntax</p> <pre>#include &lt;sdksys.h&gt; int sys_GetServersIp ( unsigned long *dns1_ip, unsigned long *dns2_ip, unsigned long *time_ip );</pre> <p>Arguments</p> <table> <tr> <td><b>dns1_ip</b></td> <td>pointer to buffer to retrieve IP address of first DNS server; if <b>dns1_ip</b> is NULL, this address is not returned</td> </tr> <tr> <td><b>dns2_ip</b></td> <td>pointer to buffer to retrieve IP address of second DNS server; if <b>dns2_ip</b> is NULL, this address is not returned</td> </tr> <tr> <td><b>time_ip</b></td> <td>pointer to buffer to retrieve IP address of time server; if <b>time_ip</b> is NULL, time server's IP is not returned</td> </tr> </table> <p>Description</p> <p>The NE-4100-P's network configuration can be set to DHCP or BOOTP mode by <b>sysc_SetIPConfig()</b>. In either of these modes, the DHCP/BOOTP server may assign IP addresses for the DNS server and time server. <b>sys_GetServersIp()</b> returns this information to the user program using the specified buffers.</p> <p>If the requested IP address is not provided, the value of 0 will be returned in the corresponding buffer.</p> <p>Return Value</p> <table> <tr> <td>0</td> <td>This function always returns 0.</td> </tr> </table>			<b>dns1_ip</b>	pointer to buffer to retrieve IP address of first DNS server; if <b>dns1_ip</b> is NULL, this address is not returned	<b>dns2_ip</b>	pointer to buffer to retrieve IP address of second DNS server; if <b>dns2_ip</b> is NULL, this address is not returned	<b>time_ip</b>	pointer to buffer to retrieve IP address of time server; if <b>time_ip</b> is NULL, time server's IP is not returned	0	This function always returns 0.
<b>dns1_ip</b>	pointer to buffer to retrieve IP address of first DNS server; if <b>dns1_ip</b> is NULL, this address is not returned									
<b>dns2_ip</b>	pointer to buffer to retrieve IP address of second DNS server; if <b>dns2_ip</b> is NULL, this address is not returned									
<b>time_ip</b>	pointer to buffer to retrieve IP address of time server; if <b>time_ip</b> is NULL, time server's IP is not returned									
0	This function always returns 0.									

<b>sys_malloc</b>	allocates memory blocks	
<p>Syntax</p> <pre>#include &lt;sdksys.h&gt; void *sys_malloc ( unsigned size );</pre> <p>Arguments</p> <p><b>size</b>                      number of elements to be allocated</p> <p>Description</p> <p>The <b>sys_malloc()</b> function allocates <b>size</b> bytes of memory. The allocated space is suitably aligned (after possible pointer coercion) for storage of any type of object.</p> <p>Note that <b>sys_malloc()</b> does NOT normally initialize the returned memory to zero bytes.</p> <p>Return Value</p> <p>The <b>sys_malloc()</b> function returns a pointer to the allocated memory if successful; otherwise a NULL pointer is returned.</p>		
<b>sys_realloc</b>	reallocate memory blocks	
<p>Syntax</p> <pre>#include &lt;sdksys.h&gt; void *sys_realloc ( void *ptr, unsigned newsize );</pre> <p>Arguments</p> <p><b>ptr</b>                          pointer to previously allocated memory block</p> <p><b>newsize</b>                      new size (bytes)</p> <p>Description</p> <p>The <b>sys_realloc()</b> function changes the size of the previously allocated memory referenced by <b>ptr</b> to <b>newsize</b> bytes. The contents of the memory are unchanged up to the lesser of the new and old sizes. If the new size is larger, the value of the newly allocated portion of the memory is undefined. If the requested memory cannot be allocated, NULL is returned and the memory referenced by <b>ptr</b> is valid and unchanged. If <b>ptr</b> is NULL, the <b>sys_realloc()</b> function behaves identically to <b>sys_malloc()</b> for the specified size.</p> <p>Return Value</p> <p>The <b>sys_realloc()</b> function returns a pointer, possibly identical to <b>ptr</b>, to the allocated memory if successful; otherwise a NULL pointer is returned. The <b>sys_realloc()</b> function always leaves the original buffer intact if an error occurs.</p>		
<b>sys_restart_system</b>	restart system	
<p>Syntax</p> <pre>#include &lt;sdksys.h&gt; void sys_restart_system ( void );</pre> <p>Arguments</p> <p>N/A</p> <p>Description</p> <p>restart system</p> <p>Return Value</p> <p>N/A</p>		

<b>sys_restart_UserAP</b>	restart user AP	
<p>Syntax</p> <pre>#include &lt;sdksys.h&gt; void sys_restart_UserAP ( void );</pre> <p>Arguments</p> <p>N/A</p> <p>Description</p> <p>restart user AP</p> <p>Return Value</p> <p>N/A</p>		
<b>sys_Set_RegisterID</b>	set application ID	
<p>Syntax</p> <pre>#include &lt;sdksys.h&gt; void sys_Set_RegisterID ( u_long id );</pre> <p>Arguments</p> <p><b>id</b> application ID; 0x00000000 to 0x7FFFFFFF only</p> <p>Description</p> <p>set the application ID; IDs between 0x80000000 and 0xFFFFFFFF are reserved for MOXA only; this function should be called as soon as application runs</p> <p>Return Value</p> <p>N/A</p>		
<b>sys_set_SerialType</b>	set async port interface signal type	
<p>Syntax</p> <pre>#include &lt;sdksys.h&gt; int sys_set_SerialType ( int port, int type );</pre> <p>Arguments</p> <p><b>port</b> async serial port number</p> <p><b>type</b> 0: RS-232 1: RS-422 2: RS-485 2-wire 3: RS-485 4-wire</p> <p>Description</p> <p>set async port interface signal type</p> <p>Return Value</p> <p>0 set OK -1 bad port -2 bad parameter (cannot set this interface type)</p>		

<b>sys_sleep_ms</b>	sleep task (milliseconds)	
<p>Syntax</p> <pre>#include &lt;sdksys.h&gt; int sys_sleep_ms ( long time_ms );</pre> <p>Arguments</p> <p><b>time_ms</b>                    sleep time (milliseconds)</p> <p>Description</p> <p>sleep task (milliseconds)</p> <p>Return Value</p> <p>This function always returns 0.</p>		
<b>sys_timeout</b>	set the timeout event service routine	
<p>Syntax</p> <pre>#include &lt;sdksys.h&gt; int sys_timeout ( void (*func)(), long time_ms );</pre> <p>Arguments</p> <p><b>func</b>                        the timeout event service routine</p> <p><b>time_ms</b>                    timeout value (milliseconds)</p> <p>Description</p> <p>set the timeout event service routine</p> <p>Return Value</p> <p>0                              no errors</p> <p>EINVAL                        the <i>isr</i> argument event function pointer is invalid</p> <p>ENOBUFS                      no resources</p>		
<b>sysc_GetDebug</b>	get debug output setting	
<p>Syntax</p> <pre>#include &lt;sdkconf.h&gt; int sysc_GetDebug ( void );</pre> <p>Arguments</p> <p>N/A</p> <p>Description</p> <p>get debug output setting</p> <p>Return Value</p> <p>0                              debug mode off</p> <p>1                              debug mode on</p>		

<b>sysc_GetGateway</b>	get IP address of gateway									
<p>Syntax</p> <pre>#include &lt;sdkconf.h&gt; u_long sysc_GetGateway ( void );</pre> <p>Arguments</p> <p>N/A</p> <p>Description</p> <p>get server gateway</p> <p>Return Value</p> <p>gateway IP address</p>										
<b>sysc_GetIP</b>	get IP address									
<p>Syntax</p> <pre>#include &lt;sdkconf.h&gt; u_long sysc_GetIP ( void );</pre> <p>Arguments</p> <p>N/A</p> <p>Description</p> <p>get server IP address</p> <p>Return Value</p> <p>the local server IP address</p>										
<b>sysc_GetIPConfig</b>	get IP configuration settings									
<p>Syntax</p> <pre>#include &lt;sdkconf.h&gt; int sysc_GetIPConfig ( void );</pre> <p>Arguments</p> <p>N/A</p> <p>Description</p> <p>get the IP configuration settings</p> <p>Return Value</p> <table> <tr> <td>0</td> <td>static IP</td> </tr> <tr> <td>1</td> <td>DHCP</td> </tr> <tr> <td>2</td> <td>DHCP &amp; BOOTP</td> </tr> <tr> <td>3</td> <td>BOOTP</td> </tr> </table>			0	static IP	1	DHCP	2	DHCP & BOOTP	3	BOOTP
0	static IP									
1	DHCP									
2	DHCP & BOOTP									
3	BOOTP									

<b>sysc_GetIPLocating</b>	get IP location setting											
<p>Syntax</p> <pre>#include &lt;sdkconf.h&gt; int sysc_GetIPLocating ( u_long *ipaddr, u_int *pno, int *time );</pre> <p>Arguments</p> <table> <tr> <td><b>ipaddr</b></td> <td>remote server IP address buffer pointer</td> </tr> <tr> <td><b>pno</b></td> <td>remote UDP port number pointer</td> </tr> <tr> <td><b>time</b></td> <td>report period time pointer</td> </tr> </table> <p>Description</p> <p>get the NE-4100-P's IP Location setting</p> <p>Return Value</p> <table> <tr> <td>0</td> <td>OK</td> </tr> <tr> <td>-2</td> <td>error argument</td> </tr> </table>			<b>ipaddr</b>	remote server IP address buffer pointer	<b>pno</b>	remote UDP port number pointer	<b>time</b>	report period time pointer	0	OK	-2	error argument
<b>ipaddr</b>	remote server IP address buffer pointer											
<b>pno</b>	remote UDP port number pointer											
<b>time</b>	report period time pointer											
0	OK											
-2	error argument											
<b>sysc_GetName</b>	get server name											
<p>Syntax</p> <pre>#include &lt;sdkconf.h&gt; int sysc_GetName ( char *name, int size );</pre> <p>Arguments</p> <table> <tr> <td><b>name</b></td> <td>server name buffer pointer</td> </tr> <tr> <td><b>size</b></td> <td>buffer size</td> </tr> </table> <p>Description</p> <p>get server name</p> <p>Return Value</p> <table> <tr> <td>&gt;=0</td> <td>the length of server name returned</td> </tr> </table>			<b>name</b>	server name buffer pointer	<b>size</b>	buffer size	>=0	the length of server name returned				
<b>name</b>	server name buffer pointer											
<b>size</b>	buffer size											
>=0	the length of server name returned											
<b>sysc_GetNetmask</b>	get netmask											
<p>Syntax</p> <pre>#include &lt;sdkconf.h&gt; u_long sysc_GetNetmask ( void );</pre> <p>Arguments</p> <p>N/A</p> <p>Description</p> <p>get server netmask</p> <p>Return Value</p> <p>the local server IP netmask</p>												

<b>sysc_GetPassword</b>	get server password													
<p>Syntax</p> <pre>#include &lt;sdkconf.h&gt; int sysc_GetPassword ( char *password, int size );</pre> <p>Arguments</p> <table> <tr> <td><b>password</b></td> <td>server password buffer pointer</td> </tr> <tr> <td><b>size</b></td> <td>buffer size</td> </tr> </table> <p>Description</p> <p>get NE-4100-P password</p> <p>Return Value</p> <table> <tr> <td><b>&gt;=0</b></td> <td>the length of password returned</td> </tr> </table>			<b>password</b>	server password buffer pointer	<b>size</b>	buffer size	<b>&gt;=0</b>	the length of password returned						
<b>password</b>	server password buffer pointer													
<b>size</b>	buffer size													
<b>&gt;=0</b>	the length of password returned													
<b>sysc_GetSerialFIFO</b>	get serial port FIFO settings													
<p>Syntax</p> <pre>#include &lt;sdkconf.h&gt; int sysc_GetSerialFIFO ( int port );</pre> <p>Arguments</p> <table> <tr> <td><b>port</b></td> <td>async serial port number</td> </tr> </table> <p>Description</p> <p>get the serial port FIFO settings</p> <p>Return Value</p> <table> <tr> <td><b>1</b></td> <td>FIFO enabled</td> </tr> <tr> <td><b>0</b></td> <td>FIFO disabled</td> </tr> <tr> <td><b>-1</b></td> <td>error port number</td> </tr> </table>			<b>port</b>	async serial port number	<b>1</b>	FIFO enabled	<b>0</b>	FIFO disabled	<b>-1</b>	error port number				
<b>port</b>	async serial port number													
<b>1</b>	FIFO enabled													
<b>0</b>	FIFO disabled													
<b>-1</b>	error port number													
<b>sysc_GetSerialInterface</b>	get serial port interface													
<p>Syntax</p> <pre>#include &lt;sdkconf.h&gt; int sysc_GetSerialInterface ( int port );</pre> <p>Arguments</p> <table> <tr> <td><b>port</b></td> <td>async serial port number</td> </tr> </table> <p>Description</p> <p>get the serial port interface</p> <p>Return Value</p> <table> <tr> <td><b>-1</b></td> <td>error port number</td> </tr> <tr> <td><b>0</b></td> <td>RS-232</td> </tr> <tr> <td><b>1</b></td> <td>RS-422</td> </tr> <tr> <td><b>2</b></td> <td>RS-485 2-wire</td> </tr> <tr> <td><b>3</b></td> <td>RS-485 4-wire</td> </tr> </table>			<b>port</b>	async serial port number	<b>-1</b>	error port number	<b>0</b>	RS-232	<b>1</b>	RS-422	<b>2</b>	RS-485 2-wire	<b>3</b>	RS-485 4-wire
<b>port</b>	async serial port number													
<b>-1</b>	error port number													
<b>0</b>	RS-232													
<b>1</b>	RS-422													
<b>2</b>	RS-485 2-wire													
<b>3</b>	RS-485 4-wire													

<b>sysc_GetSerialIoctl</b>	get serial port parameters	
Syntax		
<pre>#include &lt;sdkconf.h&gt; int sysc_GetSerialIoctl ( int port, int *baud, int *mode, int *flow );</pre>		
Arguments		
<b>port</b>	async serial port number	
<b>baud</b>	baud rate buffer pointer	
	0: 50	6: 600      12: 9600
	1: 75	7: 1200     13: 19200
	2: 110	8: 1800     14: 38400
	3: 134.5	9: 2400     15: 57600
	4: 150	10: 4800    16: 115200
	5: 300	11: 7200    17: 230400
<b>mode</b>	character mode buffer pointer	
	<b>bit_cnt</b> (bits 0-1)	
	0x00: data bit 5	
	0x01: data bit 6	
	0x02: data bit 7	
	0x03: data bit 8	
	<b>stop_bit</b> (bit 2)	
	0x00: stop bit 1	
	0x04: stop bit 1.5 or 2	
	<b>parity</b> (bits 3,4 5)	
	0x00: none parity	
	0x08: odd parity	
	0x18: even parity	
	0x28: mark parity	
	0x38: space parity	
<b>flow</b>	flow control buffer pointer	
	0: none	
	1: RTS/CTS	
	2: XON/XOFF	
	3: DTR/DSR	
Description		
get serial port parameter		
Return Value		
0	OK	
-1	error port number	



<b>sysc_SaveAndRestart</b>	save new settings and restart NE-4100-P	
<p>Syntax</p> <pre>#include &lt;sdkconf.h&gt; void sysc_SaveAndRestart ( void );</pre> <p>Arguments</p> <p>N/A</p> <p>Description</p> <p>save new settings and restart the NE-4100-P; after calling this function, the NE-4100-P will be restarted</p> <p>Return Value</p> <p>N/A</p>		
<b>sysc_SetDebug</b>	set debug output setting	
<p>Syntax</p> <pre>#include &lt;sdkconf.h&gt; int sysc_SetDebug ( int mode );</pre> <p>Arguments</p> <p><b>mode</b>                    0: off                              1: on</p> <p>Description</p> <p>set debug output setting</p> <p>Return Value</p> <p>0                            OK -2                           error argument</p>		
<b>sysc_SetGateway</b>	set gateway address	
<p>Syntax</p> <pre>#include &lt;sdkconf.h&gt; int sysc_SetGateway ( u_long ipaddr );</pre> <p>Arguments</p> <p><b>ipaddr</b>                    new gateway IP address</p> <p>Description</p> <p>set gateway</p> <p>Return Value</p> <p>0                            OK -2                           error argument</p>		

<b>sysc_SetIP</b>	set IP address	
<p>Syntax</p> <pre>#include &lt;sdkconf.h&gt; int sysc_SetIP ( u_long ipaddr );</pre> <p>Arguments</p> <p><b>ipaddr</b>                    new local server IP address</p> <p>Description</p> <p>set IP address</p> <p>Return Value</p> <p>0                            OK</p> <p>-2                           error argument</p>		
<b>sysc_SetIPConfig</b>	define how IP address, netmask and gateway are obtained	
<p>Syntax</p> <pre>#include &lt;sdkconf.h&gt; int sysc_SetIPConfig ( int type );</pre> <p>Arguments</p> <p><b>type</b>                        IP configuration</p> <p>                              0: static IP</p> <p>                              1: DHCP</p> <p>                              2: DHCP &amp; BOOTP</p> <p>                              3: BOOTP</p> <p>Description</p> <p>define how IP address, netmask and gateway are obtained</p> <p>Return Value</p> <p>0                            OK</p> <p>-2                           error argument</p>		
<b>sysc_SetIPLocating</b>	set IP location function	
<p>Syntax</p> <pre>#include &lt;sdkconf.h&gt; int sysc_SetIPLocating ( u_long ipaddr, int pno, int time );</pre> <p>Arguments</p> <p><b>ipaddr</b>                    IP address of IP location remote server; set 0.0.0.0 to disable this function</p> <p><b>pno</b>                         UDP port number of IP location remote server</p> <p><b>time</b>                        report period time (seconds)</p> <p>Description</p> <p>set IP location function</p> <p>Return Value</p> <p>0                            OK</p> <p>-2                           error argument</p>		

<b>sysc_SetName</b>	set server name	
<p>Syntax</p> <pre>#include &lt;sdkconf.h&gt; int sysc_SetName ( char *name );</pre> <p>Arguments</p> <p><b>name</b>                    new server name</p> <p>Description</p> <p>set server name</p> <p>Return Value</p> <p>0                            OK</p>		
<b>sysc_SetNetmask</b>	set netmask	
<p>Syntax</p> <pre>#include &lt;sdkconf.h&gt; int sysc_SetNetmask ( u_long netmask );</pre> <p>Arguments</p> <p><b>netmask</b>                  new local server netmask</p> <p>Description</p> <p>set NE-4100-P netmask</p> <p>Return Value</p> <p>0                            OK</p>		
<b>sysc_SetPassword</b>	set password	
<p>Syntax</p> <pre>#include &lt;sdkconf.h&gt; int sysc_SetPassword ( char *password );</pre> <p>Arguments</p> <p><b>password</b>                  new server password</p> <p>Description</p> <p>set NE-4100-P password</p> <p>Return Value</p> <p>0                            OK</p>		

<code>sysc_SetSerialFIFO</code>	set serial port FIFO settings									
<p>Syntax</p> <pre>#include &lt;sdkconf.h&gt; int sysc_SetSerialFIFO ( int port, int mode );</pre>										
<p>Arguments</p> <table> <tr> <td><code>port</code></td> <td>async serial port number</td> </tr> <tr> <td><code>mode</code></td> <td>FIFO mode</td> </tr> <tr> <td></td> <td>0: disable</td> </tr> <tr> <td></td> <td>1: enable</td> </tr> </table>			<code>port</code>	async serial port number	<code>mode</code>	FIFO mode		0: disable		1: enable
<code>port</code>	async serial port number									
<code>mode</code>	FIFO mode									
	0: disable									
	1: enable									
<p>Description</p> <p>set serial port FIFO settings</p>										
<p>Return Value</p> <table> <tr> <td>0</td> <td>OK</td> </tr> <tr> <td>-1</td> <td>error port number</td> </tr> <tr> <td>-2</td> <td>error argument</td> </tr> </table>			0	OK	-1	error port number	-2	error argument		
0	OK									
-1	error port number									
-2	error argument									

<code>sysc_SetSerialInterface</code>	set the serial port interface													
<p>Syntax</p> <pre>#include &lt;sdkconf.h&gt; int sysc_SetSerialInterface ( int port, int type );</pre>														
<p>Arguments</p> <table> <tr> <td><code>port</code></td> <td>async serial port number</td> </tr> <tr> <td><code>type</code></td> <td>serial interface type</td> </tr> <tr> <td></td> <td>0: RS-232</td> </tr> <tr> <td></td> <td>1: RS-422</td> </tr> <tr> <td></td> <td>2: RS-485 2-wire</td> </tr> <tr> <td></td> <td>3: RS-485 4-wire</td> </tr> </table>			<code>port</code>	async serial port number	<code>type</code>	serial interface type		0: RS-232		1: RS-422		2: RS-485 2-wire		3: RS-485 4-wire
<code>port</code>	async serial port number													
<code>type</code>	serial interface type													
	0: RS-232													
	1: RS-422													
	2: RS-485 2-wire													
	3: RS-485 4-wire													
<p>Description</p> <p>set serial port interface</p>														
<p>Return Value</p> <table> <tr> <td>0</td> <td>OK</td> </tr> <tr> <td>-1</td> <td>error port number</td> </tr> <tr> <td>-2</td> <td>error argument</td> </tr> </table>			0	OK	-1	error port number	-2	error argument						
0	OK													
-1	error port number													
-2	error argument													

<code>sysc_SetSerialIoctl</code>	set serial port parameter																		
Syntax																			
<pre>#include &lt;sdkconf.h&gt; int sysc_SetSerialIoctl ( int port, int baud, int mode, int flow );</pre>																			
Arguments																			
<b>port</b>	async serial port number																		
<b>baud</b>	<table> <tr> <td>0: 50</td> <td>6: 600</td> <td>12: 9600</td> </tr> <tr> <td>1: 75</td> <td>7: 1200</td> <td>13: 19200</td> </tr> <tr> <td>2: 110</td> <td>8: 1800</td> <td>14: 38400</td> </tr> <tr> <td>3: 134.5</td> <td>9: 2400</td> <td>15: 57600</td> </tr> <tr> <td>4: 150</td> <td>10: 4800</td> <td>16: 115200</td> </tr> <tr> <td>5: 300</td> <td>11: 7200</td> <td>17: 230400</td> </tr> </table>	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
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																	
<b>mode</b>	<b>bit_cnt</b> OR <b>stop_bit</b> OR <b>parity</b> <b>bit_cnt</b> (bits 0-1) 0x00: data bit 5 0x01: data bit 6 0x02: data bit 7 0x03: data bit 8 <b>stop_bit</b> (bit 2) 0x00: stop bit 1 0x04: stop bits 1.5 or 2 <b>parity</b> (bits 3- 5) 0x00: no parity 0x08: odd parity 0x18: even parity 0x28: mark parity 0x38: space parity																		
<b>flow</b>	flow control 0: none 1: RTS/CTS 2: XON/XOFF 3: DTR/DSR																		
Description																			
set serial port parameters																			
Return Value																			
0	OK																		
-1	error port number																		
-2	error argument																		

<b>sysc_SetToDefault</b>	set to default value	
Syntax		
<pre>#include &lt;sdkconf.h&gt; void sysc_SetToDefault ( void );</pre>		
Arguments		
N/A		
Description		
restore all configurations back to factory default value		
Return Value		
N/A		

## Flash ROM Access Library Reference

<b>flash_erase</b>	erase flash ROM	
Syntax		
<pre>#include &lt;sdkflash.h&gt; int flash_erase ( void );</pre>		
Arguments		
N/A		
Description		
erase flash ROM		
Return Value		
0	OK	
-1	fail	

<b>flash_length</b>	get current data length of flash ROM	
Syntax		
<pre>#include &lt;sdkflash.h&gt; long flash_length ( void );</pre>		
Arguments		
N/A		
Description		
get current data length of flash ROM		
Return Value		
>=0	current data length of flash ROM; max length is 163840 (160 KB)	
0	value after calling <code>sys_flash_erase()</code>	

<b>flash_read</b>	read data from flash ROM	
Syntax		
<pre>#include &lt;sdkflash.h&gt; long flash_read ( long offset, char *buffer, long size );</pre>		
Arguments		
<b>offset</b>	indicates the point at which the function starts reading the buffer, measured in bytes	
<b>buffer</b>	read buffer pointer	
<b>size</b>	buffer size	
Description		
read data from the flash ROM		
Return Value		
<b>&gt;=0</b>	read data size	
<b>-1</b>	read failed	

<b>flash_write</b>	write data to flash ROM	
Syntax		
<pre>#include &lt;sdkflash.h&gt; long flash_write ( char *buffer, long size );</pre>		
Arguments		
<b>buffer</b>	write data buffer pointer	
<b>size</b>	write data size, from 1 to 163840	
Description		
write data to the flash ROM; allows specific portions of flash ROM to be written		
Return Value		
<b>&gt;0</b>	write length	
<b>-1</b>	write failed	
<b>-2</b>	need to erase flash ROM first	

<b>sys_FlashErase</b>	erase flash ROM	
Syntax		
<pre>#include &lt;sdkflash.h&gt; int sys_FlashErase ( int bank );</pre>		
Arguments		
<b>bank</b>	memory bank, from 0 to 4	
Description		
erase flash ROM		
Return Value		
<b>-1</b>	fail	
<b>-2</b>	argument error	
<b>0</b>	OK	

<b>sys_FlashLength</b>	get current data length of flash ROM	
<p>Syntax</p> <pre>#include &lt;sdkflash.h&gt; long sys_FlashLength ( int bank );</pre> <p>Arguments</p> <p><b>bank</b>                      memory bank, from 0 to 4</p> <p>Description</p> <p>get current data length of flash ROM</p> <p>Return Value</p> <p><b>&gt;=0</b>                      current data length of flash ROM; max length is 32768 (32KB)  <b>0</b>                              value after calling <b>sys_FlashErase()</b>  <b>-2</b>                              argument error</p>		
<b>sys_FlashRead</b>	read data to flash ROM	
<p>Syntax</p> <pre>#include &lt;sdkflash.h&gt; long sys_FlashRead ( int bank, long offset, char * buffer, long size );</pre> <p>Arguments</p> <p><b>bank</b>                      memory bank, from 0 to 4  <b>offset</b>                      start read offset from bank begin  <b>buffer</b>                      read buffer pointer  <b>size</b>                        write data size, from 1 to 32768</p> <p>Description</p> <p>read data to flash ROM; may be used to read specific segment of flash ROM rather than the entire flash ROM</p> <p>Return Value</p> <p><b>&gt;=0</b>                      size of data read  <b>-1</b>                              read fail  <b>-2</b>                              argument error</p>		
<b>sys_FlashWrite</b>	write data to flash ROM	
<p>Syntax</p> <pre>#include &lt;sdkflash.h&gt; long sys_FlashWrite ( int bank, char *buffer, long size );</pre> <p>Arguments</p> <p><b>bank</b>                      memory bank, from 0 to 4  <b>buffer</b>                      write data buffer pointer  <b>size</b>                        write data size, from 1 to 32768</p> <p>Description</p> <p>write data to flash ROM</p> <p>Return Value</p> <p><b>&gt;0</b>                        length of data written  <b>-1</b>                              write fail  <b>-2</b>                              argument error  <b>-3</b>                              need to erase flash ROM first</p>		



## Debug Library Reference

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

<b>dbg_printf</b>	print formatted output to debug output stream					
Syntax <pre>#include &lt;sdkdbg.h&gt; int dbg_printf ( char *format, [, argument]... );</pre>						
Arguments <table> <tr> <td><b>format</b></td> <td>format control</td> </tr> <tr> <td><b>argument</b></td> <td>optional arguments</td> </tr> </table>			<b>format</b>	format control	<b>argument</b>	optional arguments
<b>format</b>	format control					
<b>argument</b>	optional arguments					
Description print formatted output to debug output stream						
Return Value This function returns the number of characters printed, or a negative value if an error occurs.						

<b>dbg_put_block</b>	print out a block of data for debugging					
Syntax <pre>#include &lt;sdkdbg.h&gt; int dbg_put_block ( char *buf, int len );</pre>						
Arguments <table> <tr> <td><b>buf</b></td> <td>the print out debugging data buffer pointer</td> </tr> <tr> <td><b>len</b></td> <td>length of the debugging data buffer</td> </tr> </table>			<b>buf</b>	the print out debugging data buffer pointer	<b>len</b>	length of the debugging data buffer
<b>buf</b>	the print out debugging data buffer pointer					
<b>len</b>	length of the debugging data buffer					
Description print out a block of data for debugging						
Return Value This function returns the length of data that is printed out.						

<b>dbg_put_ch</b>	print out a character for debugging			
Syntax <pre>#include &lt;sdkdbg.h&gt; int dbg_put_ch ( char ch );</pre>				
Arguments <table> <tr> <td><b>ch</b></td> <td>the character value that will be printed out</td> </tr> </table>			<b>ch</b>	the character value that will be printed out
<b>ch</b>	the character value that will be printed out			
Description print out a character for debugging				
Return Value This function returns the length of data that is printed out.				

<b>dbg_put_doubleword</b>	print out a 4-byte unsigned long value for debugging	
<p>Syntax</p> <pre>#include &lt;sdkdbg.h&gt; int dbg_put_doubleword ( unsigned long value );</pre> <p>Arguments</p> <p><b>value</b>                      the printed out unsigned long value</p> <p>Description</p> <p>print out a 4-byte unsigned long value for debugging</p> <p>Return Value</p> <p>This function returns the length of data that is printed out.</p>		
<b>dbg_put_doubleword_hex</b>	print out a 4-byte unsigned long value with HEX format for debugging	
<p>Syntax</p> <pre>#include &lt;sdkdbg.h&gt; int dbg_put_doubleword_hex ( unsigned long value );</pre> <p>Arguments</p> <p><b>value</b>                      the printed out unsigned long value</p> <p>Description</p> <p>print out a 4-byte unsigned long value with HEX format for debugging</p> <p>Return Value</p> <p>This function returns the length of data that is printed out.</p>		
<b>dbg_put_IP</b>	print out an IP address in the a.b.c.d format for debugging	
<p>Syntax</p> <pre>#include &lt;sdkdbg.h&gt; int dbg_put_IP ( unsigned long ipaddr );</pre> <p>Arguments</p> <p><b>ipaddr</b>                      the printed out Internet host's IP address</p> <p>Description</p> <p>print out an IP address in the a.b.c.d format for debugging</p> <p>Return Value</p> <p>This function returns the length of data that is printed out.</p>		

<b>dbg_put_string</b>	print out a string for debugging	
<p>Syntax</p> <pre>#include &lt;sdkdbg.h&gt; int dbg_put_string ( char *buf );</pre> <p>Arguments</p> <p><b>buf</b>                      the printed out debugging data buffer's pointer</p> <p>Description</p> <p>print out a string for debugging</p> <p>Return Value</p> <p>This function returns the length of data that is printed out.</p>		
<b>dbg_put_word</b>	print out a 2-byte unsigned integer value for debugging	
<p>Syntax</p> <pre>#include &lt;sdkdbg.h&gt; int dbg_put_word ( unsigned short value );</pre> <p>Arguments</p> <p><b>value</b>                      the printed out unsigned short value</p> <p>Description</p> <p>print out a 2-byte unsigned integer value for debugging</p> <p>Return Value</p> <p>This function returns the length of data that is printed out.</p>		
<b>dbg_put_word_hex</b>	print out a 2-byte unsigned integer value with HEX format for debugging	
<p>Syntax</p> <pre>#include &lt;sdkdbg.h&gt; int dbg_put_word_hex ( unsigned short value );</pre> <p>Arguments</p> <p><b>value</b>                      the print out unsigned short value</p> <p>Description</p> <p>print out a 2-byte unsigned integer value with HEX format for debugging</p> <p>Return Value</p> <p>This function returns the length of data that is printed out.</p>		

## DIO Library Reference

<b>DIO_ControlSingleIO</b>	set output channel to high or low state									
<p>Syntax</p> <pre>#include &lt;sdkdio.h&gt; int DIO_ControlSingleIO ( int io, int highlow );</pre> <p>Arguments</p> <table> <tr> <td><b>io</b></td> <td>I/O number</td> </tr> <tr> <td><b>highlow</b></td> <td>0: set the output state to low != 0: set the output state to high</td> </tr> </table> <p>Description</p> <p>set output channel to high or low state</p> <p>Return Value</p> <table> <tr> <td>0</td> <td>OK</td> </tr> <tr> <td>-2</td> <td>fail</td> </tr> </table>			<b>io</b>	I/O number	<b>highlow</b>	0: set the output state to low != 0: set the output state to high	0	OK	-2	fail
<b>io</b>	I/O number									
<b>highlow</b>	0: set the output state to low != 0: set the output state to high									
0	OK									
-2	fail									
<b>DIO_GetSingleIO</b>	get mode (input or output) of DIO channel									
<p>Syntax</p> <pre>#include &lt;sdkdio.h&gt; int DIO_GetSingleIO ( int io, int *mode );</pre> <p>Arguments</p> <table> <tr> <td><b>io</b></td> <td>I/O number</td> </tr> <tr> <td><b>mode</b></td> <td>pointer to buffer to retrieve the I/O mode *mode = 0 for input mode, 1 for output mode</td> </tr> </table> <p>Description</p> <p>get mode (input or output) of DIO channel</p> <p>Return Value</p> <table> <tr> <td>0</td> <td>OK</td> </tr> <tr> <td>-2</td> <td>fail</td> </tr> </table>			<b>io</b>	I/O number	<b>mode</b>	pointer to buffer to retrieve the I/O mode *mode = 0 for input mode, 1 for output mode	0	OK	-2	fail
<b>io</b>	I/O number									
<b>mode</b>	pointer to buffer to retrieve the I/O mode *mode = 0 for input mode, 1 for output mode									
0	OK									
-2	fail									
<b>DIO_GetSingleIOStatus</b>	get current state of DIO channel (high or low)									
<p>Syntax</p> <pre>#include &lt;sdkdio.h&gt; int DIO_GetSingleIOStatus ( int io, int *highlow );</pre> <p>Arguments</p> <table> <tr> <td><b>io</b></td> <td>I/O number</td> </tr> <tr> <td><b>highlow</b></td> <td>pointer to buffer to retrieve state of DIO channel *highlow = 0 for low state, 1 for high state</td> </tr> </table> <p>Description</p> <p>get current state of DIO channel (high or low)</p> <p>Return Value</p> <table> <tr> <td>0</td> <td>OK</td> </tr> <tr> <td>-2</td> <td>fail</td> </tr> </table>			<b>io</b>	I/O number	<b>highlow</b>	pointer to buffer to retrieve state of DIO channel *highlow = 0 for low state, 1 for high state	0	OK	-2	fail
<b>io</b>	I/O number									
<b>highlow</b>	pointer to buffer to retrieve state of DIO channel *highlow = 0 for low state, 1 for high state									
0	OK									
-2	fail									

<b>DIO_SetSingleIO</b>	set DIO channel to input or output mode									
<p>Syntax</p> <pre>#include &lt;sdkdio.h&gt; int DIO_SetSingleIO ( int io, int mode );</pre> <p>Arguments</p> <table> <tr> <td><b>io</b></td> <td>I/O number</td> </tr> <tr> <td><b>mode</b></td> <td>0: set to input !=0: set to output</td> </tr> </table> <p>Description</p> <p>set DIO channel to input or output mode</p> <p>Return Value</p> <table> <tr> <td>0</td> <td>OK</td> </tr> <tr> <td>-2</td> <td>fail</td> </tr> </table>			<b>io</b>	I/O number	<b>mode</b>	0: set to input !=0: set to output	0	OK	-2	fail
<b>io</b>	I/O number									
<b>mode</b>	0: set to input !=0: set to output									
0	OK									
-2	fail									
<b>Scf_getSDioMode</b>	get initial mode (input or output) for DIO channel									
<p>Syntax</p> <pre>#include &lt;sdkdio.h&gt; int Scf_getSDioMode ( int io, int *mode );</pre> <p>Arguments</p> <table> <tr> <td><b>io</b></td> <td>I/O number</td> </tr> <tr> <td><b>mode</b></td> <td>pointer to buffer to retrieve the initial I/O mode *mode = 0 for input mode, 1 for output mode</td> </tr> </table> <p>Description</p> <p>get initial I/O mode (input or output) for DIO channel</p> <p>Return Value</p> <table> <tr> <td>0</td> <td>OK</td> </tr> <tr> <td>-2</td> <td>fail</td> </tr> </table>			<b>io</b>	I/O number	<b>mode</b>	pointer to buffer to retrieve the initial I/O mode *mode = 0 for input mode, 1 for output mode	0	OK	-2	fail
<b>io</b>	I/O number									
<b>mode</b>	pointer to buffer to retrieve the initial I/O mode *mode = 0 for input mode, 1 for output mode									
0	OK									
-2	fail									
<b>Scf_getSDioState</b>	get initial output state (high or low) for output channel									
<p>Syntax</p> <pre>#include &lt;sdkdio.h&gt; int Scf_getSDioState ( int io, int *highlow );</pre> <p>Arguments</p> <table> <tr> <td><b>io</b></td> <td>I/O number</td> </tr> <tr> <td><b>highlow</b></td> <td>pointer to buffer to retrieve initial I/O state *highlow = 0 for low state, 1 for high state</td> </tr> </table> <p>Description</p> <p>get initial output state (high or low) for output channel; this setting takes effect only if the initial I/O mode is set to output mode.</p> <p>Return Value</p> <table> <tr> <td>0</td> <td>OK</td> </tr> <tr> <td>-2</td> <td>fail</td> </tr> </table>			<b>io</b>	I/O number	<b>highlow</b>	pointer to buffer to retrieve initial I/O state *highlow = 0 for low state, 1 for high state	0	OK	-2	fail
<b>io</b>	I/O number									
<b>highlow</b>	pointer to buffer to retrieve initial I/O state *highlow = 0 for low state, 1 for high state									
0	OK									
-2	fail									

<b>Scf_setSDioMode</b>	set initial mode of DIO channel to input or output									
<p>Syntax</p> <pre>#include &lt;sdkdio.h&gt; int Scf_setSDioMode ( int io, int mode );</pre> <p>Arguments</p> <table> <tr> <td><b>io</b></td> <td>I/O number</td> </tr> <tr> <td><b>mode</b></td> <td>0: set initial mode to input != 0: set initial mode to output</td> </tr> </table> <p>Description</p> <p>set the initial mode of DIO channel to input or output; this function defines the channel's initial mode when NE-4100-P boots up</p> <p>Return Value</p> <table> <tr> <td>0</td> <td>OK</td> </tr> <tr> <td>-2</td> <td>fail</td> </tr> </table>			<b>io</b>	I/O number	<b>mode</b>	0: set initial mode to input != 0: set initial mode to output	0	OK	-2	fail
<b>io</b>	I/O number									
<b>mode</b>	0: set initial mode to input != 0: set initial mode to output									
0	OK									
-2	fail									

<b>Scf_setSDioState</b>	set initial state of output channel to high or low									
<p>Syntax</p> <pre>#include &lt;sdkdio.h&gt; int Scf_setSDioState ( int io, int highlow );</pre> <p>Arguments</p> <table> <tr> <td><b>io</b></td> <td>I/O number</td> </tr> <tr> <td><b>highlow</b></td> <td>0: set the initial output state to low != 0: set the initial output state to high</td> </tr> </table> <p>Description</p> <p>set initial state of output channel to high or low</p> <p>Return Value</p> <table> <tr> <td>0</td> <td>OK</td> </tr> <tr> <td>-2</td> <td>fail</td> </tr> </table>			<b>io</b>	I/O number	<b>highlow</b>	0: set the initial output state to low != 0: set the initial output state to high	0	OK	-2	fail
<b>io</b>	I/O number									
<b>highlow</b>	0: set the initial output state to low != 0: set the initial output state to high									
0	OK									
-2	fail									

## Thread Control Library Reference

<b>sys_ThreadClose</b>	close a thread							
<p>Syntax</p> <pre>#include &lt;sdktask.h&gt; int sys_ThreadClose ( unsigned long threadID );</pre> <p>Arguments</p> <table> <tr> <td><b>threadID</b></td> <td>thread ID returned from <b>sys_ThreadCreate()</b></td> </tr> </table> <p>Description</p> <p>close a thread</p> <p>Return Value</p> <table> <tr> <td>0</td> <td>OK</td> </tr> <tr> <td>-1</td> <td>error threadID</td> </tr> </table>			<b>threadID</b>	thread ID returned from <b>sys_ThreadCreate()</b>	0	OK	-1	error threadID
<b>threadID</b>	thread ID returned from <b>sys_ThreadCreate()</b>							
0	OK							
-1	error threadID							

<b>sys_ThreadCreate</b>	create a thread	
Syntax		
<pre>#include &lt;sdktask.h&gt; int sys_ThreadCreate ( void (*ap) ( unsigned long arg ), char *stack_ptr, long stack_size, unsigned long parameter, int createFlags, unsigned long *threadIDp );</pre>		
Arguments		
<b>ap</b>	this thread entry pointer	
<b>stack_ptr</b>	this thread stack pointer	
<b>stack_size</b>	this thread stack size	
<b>parameter</b>	the argument passing to <b>ap()</b>	
<b>createFlags</b>	currently not used, must be set to 0	
<b>threadIDp</b>	pointer to a buffer to return the new thread ID	
Description		
create a thread		
Return Value		
0	OK	
-1	no resource	
-2	error argument	

<b>sys_ThreadResume</b>	resume a thread	
Syntax		
<pre>#include &lt;sdktask.h&gt; int sys_ThreadResume ( unsigned long threadID );</pre>		
Arguments		
<b>threadID</b>	thread ID returned from <b>sys_ThreadCreate()</b>	
Description		
resume a thread		
Return Value		
0	OK	
-1	error threadID	

<b>sys_ThreadState</b>	get a thread state	
Syntax		
<pre>#include &lt;sdktask.h&gt; int sys_ThreadState ( unsigned long threadID );</pre>		
Arguments		
<b>threadID</b>	thread ID returned from <b>sys_ThreadCreate()</b>	
Description		
get a thread state		
Return Value		
1	suspend	
0	running	
-1	error threadID	

<b>sys_ThreadSuspend</b>	suspend a thread	
Syntax		
<pre>#include &lt;sdktask.h&gt; int sys_ThreadSuspend ( unsigned long threadID );</pre>		
Arguments		
<b>threadID</b>	thread ID returned from <code>sys_ThreadCreate()</code>	
Description		
suspend a thread		
Return Value		
0	OK	
-1	error threadID	

## Time Server Library Reference

<b>sys_GetLocalTime</b>	get local time	
Syntax		
<pre>#include &lt;sdksys.h&gt; int sys_GetLocalTime ( struct tm_local *tm );</pre>		
Arguments		
<b>tm</b>	local time information	
Description		
get local time		
Return Value		
0	OK	
-1	fail	

<b>sys_SetLocalTime</b>	set local time	
Syntax		
<pre>#include &lt;sdksys.h&gt; int sys_SetLocalTime ( struct tm_local *tm );</pre>		
Arguments		
<b>tm</b>	time information to be set	
Description		
set local time		
Return Value		
0	OK	
-1	fail	



<b>sysc_getTimeServer</b>	get IP address of time server									
<p>Syntax</p> <pre>#include &lt;sdkconf.h&gt; int sysc_getTimeServer ( char *buffer, int bufsize );</pre> <p>Arguments</p> <table> <tr> <td><b>buffer</b></td> <td>pointer to the buffer to retrieve the time server address</td> </tr> <tr> <td><b>bufsize</b></td> <td>the size of <b>buffer</b> (bytes)</td> </tr> </table> <p>Description</p> <p>get IP address of time server used to synchronize the system time</p> <p>Return Value</p> <table> <tr> <td><b>&gt;= 0</b></td> <td>number of bytes placed in buffer</td> </tr> <tr> <td><b>-1</b></td> <td>fail</td> </tr> </table>			<b>buffer</b>	pointer to the buffer to retrieve the time server address	<b>bufsize</b>	the size of <b>buffer</b> (bytes)	<b>&gt;= 0</b>	number of bytes placed in buffer	<b>-1</b>	fail
<b>buffer</b>	pointer to the buffer to retrieve the time server address									
<b>bufsize</b>	the size of <b>buffer</b> (bytes)									
<b>&gt;= 0</b>	number of bytes placed in buffer									
<b>-1</b>	fail									
<b>sysc_getTimeZone</b>	get the time offset used by time synchronization									
<p>Syntax</p> <pre>#include &lt;sdkconf.h&gt; long sysc_getTimeZone ( void );</pre> <p>Arguments</p> <p>N/A</p> <p>Description</p> <p>This function will retrieve the time offset from local time zone to UTC. The offset is used in time synchronization. The returned value is the time offset to UTC (seconds). For example, a GMT +8:00 time zone has an offset of 28800 seconds.</p> <p>Return Value</p> <table> <tr> <td><b>&gt;= 0</b></td> <td>time zone index number</td> </tr> <tr> <td><b>-1</b></td> <td>fail</td> </tr> </table>			<b>&gt;= 0</b>	time zone index number	<b>-1</b>	fail				
<b>&gt;= 0</b>	time zone index number									
<b>-1</b>	fail									
<b>sysc_getTZoneIndex</b>	get the time zone of local system									
<p>Syntax</p> <pre>#include &lt;sdkconf.h&gt; int sysc_getTZoneIndex ( void );</pre> <p>Arguments</p> <p>N/A</p> <p>Description</p> <p>retrieve the time zone of local system; all time zones are listed later in this chapter</p> <p>Return Value</p> <table> <tr> <td><b>&gt;= 0</b></td> <td>time zone index number</td> </tr> <tr> <td><b>-1</b></td> <td>fail</td> </tr> </table>			<b>&gt;= 0</b>	time zone index number	<b>-1</b>	fail				
<b>&gt;= 0</b>	time zone index number									
<b>-1</b>	fail									

<b>sysc_setTimeServer</b>	set IP address of time server that is used to synchronize the system time									
<p>Syntax</p> <pre>#include &lt;sdkconf.h&gt; int sysc_setTimeServer ( char *buffer, int bufsize );</pre> <p>Arguments</p> <table> <tr> <td><b>buffer</b></td> <td>the new time server address</td> </tr> <tr> <td><b>bufsize</b></td> <td>the length of server address in buffer (bytes)</td> </tr> </table> <p>Description</p> <p>The NE-4100-P can synchronize its system time with a remote NTP server. To enable this function, the NTP server address must be specified. A software timer is used to simulate a real time clock. NTP is used to synchronize the date and time of the internal clock with time server. If time information cannot be obtained due to network trouble, the system time will be set to Jan.1, 2000.</p> <p>Return Value</p> <table> <tr> <td><b>&gt;= 0</b></td> <td>number of bytes copied from buffer to system configuration</td> </tr> <tr> <td><b>-1</b></td> <td>fail</td> </tr> </table>			<b>buffer</b>	the new time server address	<b>bufsize</b>	the length of server address in buffer (bytes)	<b>&gt;= 0</b>	number of bytes copied from buffer to system configuration	<b>-1</b>	fail
<b>buffer</b>	the new time server address									
<b>bufsize</b>	the length of server address in buffer (bytes)									
<b>&gt;= 0</b>	number of bytes copied from buffer to system configuration									
<b>-1</b>	fail									
<b>sysc_setTimeZone</b>	set the time offset used for time synchronization									
<p>Syntax</p> <pre>#include &lt;sdkconf.h&gt; void sysc_setTimeZone ( long tz );</pre> <p>Arguments</p> <table> <tr> <td><b>tz</b></td> <td>the time offset from local time zone to UTC</td> </tr> </table> <p>Description</p> <p>This function will set the time offset from local time zone to UTC. The offset will be used in time synchronization. The offset is measured in seconds. For example, set this value to 28800 for a GMT +8:00 time zone.</p> <p>Return Value</p> <p>N/A</p>			<b>tz</b>	the time offset from local time zone to UTC						
<b>tz</b>	the time offset from local time zone to UTC									
<b>sysc_setTZoneIndex</b>	set the time zone of local system									
<p>Syntax</p> <pre>#include &lt;sdkconf.h&gt; void sysc_setTZoneIndex ( int index );</pre> <p>Arguments</p> <table> <tr> <td><b>index</b></td> <td>time zone index number</td> </tr> </table> <p>Description</p> <p>This function will set the time zone of local system. All time zones are listed later in this chapter. Note that time zone indicated by <b>index</b> is used for human readability and takes no effect when synchronizing the local time with the time server. You must call <b>sysc_setTimeZone()</b> to set the actual time offset.</p> <p>Return Value</p> <p>N/A</p>			<b>index</b>	time zone index number						
<b>index</b>	time zone index number									

## Time Zone Offsets Index

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

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

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

# A

## External Function Calls

---

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

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

Note that to use these string functions, you must link to the `c1.lib` library file, with a `tlink` command similar to the one shown below:

```
%path:>tlink /t /s c0sdk+ap, ap, ap, moxa_sdk+c:\tc\lib\c1.lib
```



### ATTENTION

You must use the complete path to link to the `c1.lib` library file.

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



### ATTENTION

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

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

# B

## Service Information

---

This appendix contains information on how to obtain information or service from MOXA for your NE-4100-P and other MOXA products. The following topics are covered:

- MOXA Internet Services**
  - Technical Support E-mail Address
  - Website for Product Information
- Problem Report Form**
- Product Return Procedure**

## MOXA Internet Services

Customer satisfaction is our top priority. 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:

- Technical Support E-mail Address  
[support@moxa.com](mailto:support@moxa.com)
- Website for Product Information  
<http://www.moxa.com>





## Product Return Procedure

For product repair, exchange, or refund, the customer must:

- Provide evidence of original purchase.
- Obtain a Product Return Agreement (PRA) from the sales representative or dealer.
- Fill out the Problem Report Form (PRF). Include as much detail as possible for a shorter product repair time.
- Carefully pack the product in an anti-static package and send it, pre-paid, to the dealer. The PRA should be visible on the outside of the package and should include a description of the problem along with the return address and telephone number.