NPort PCG SDK 2 API Reference

Third Edition, July 2005

www.moxa.com/product

MOXA Technologies Co., Ltd.
Tel: +886-2-8919-1230
Fax: +886-2-8919-1231
Web: www.moxa.com

MOXA Technical Support
Worldwide: support@moxa.com

NPort PCG SDK 2 API Reference

MOXA Internet Services

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

The following services are provided:

E-mail for technical support

e-mail address: support@moxa.com.....(Worldwide)
support@usa.moxa.com .....(the Americas)

Web site for product information

web addresses: http://www.moxa.com
or
http://www.moxa.com.tw

Table of Contents

1

Overview

The purpose of this *MOXA NPort PCG SDK 2 API Reference* is to give MOXA NPort PCG programmers a complete reference guide to the various function calls that are available. You may also refer to the companion guide, *NPort PCG SDK 2 Programmer's Guide*. The acronym **API**, which stands for **Application Programming Interface**, is used throughout this document. An API includes the necessary function calls and linking libraries.

The following topic is covered in this chapter:

- **MOXA API Quick Reference**

MOXA API Quick Reference

In this document, the SDK API functions are displayed in the format shown below.

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

A specific example is shown below. The function **sio_oqueue()** is from the SDK API Serial I/O library. This function reports the amount of data that is waiting to be transmitted out through the serial port.

sio_oqueue	Get the length of the data in the output buffers	**Port Status**
C Format		
Syntax	**#include <sdksio.h>**	
	long sio_oqueue (int *port*);	
Arguments	*port*	Async serial port number
Description	Get the length of the data in the system's output buffer and the driver's output buffer that has not yet been transmitted.	
Return Value	>= 0	length of data (in bytes) still remaining in the driver's output buffer.
	SIO_BADPORT	Port number is invalid.

2

SDK API Overview

SDK stands for Software Development Kit. The SDK includes the SDK APIs, SDK Utilities (the Windows utilities used by the programmer to communicate with the NPort PCG), and several detailed example programs. You may also refer to the companion guide, *NPort PCG SDK Programmer's Guide*, for additional information about using the utility.

In order to make the SDK library easier to use, the function calls are divided by attribute into several groups. The groups are 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, and Memory Management API.

In this chapter, we give a brief introduction to all function calls for each of the groups to give programmers a good overview of all of the APIs. To see detailed usage information for each API, refer to the following sections:

The following topics are covered in this chapter:

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

Serial I/O API

In this section, we categorize the serial I/O library routines according to their function (Port Control, Data Input, Data Output, Port Status Inquiry, Event Control, and Miscellaneous). See Chapter 3 for a more detailed description of these functions.

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

Port Control This category includes functions to open serial ports, set communication parameters, and control signal lines.	
Function Name	**Description**
sio_open	Enable a serial port for data transmitting/receiving.
sio_close	Disable a serial port for data transmitting/receiving.
sio_ioctl	Control the settings of the serial port's I/O control register.
sio_flowctrl	Set hardware and/or software flow control.
sio_flush	Flush the driver's input/output buffer.
sio_DTR	Set the DTR state of a port.
sio_RTS	Set the RTS state of a port.
sio_lctrl	Set both the DTR and RTS states.
sio_baud	Set baud rate using the actual speed value.

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

Data Output This category includes functions to write data to the serial port.	
Function Name	**Description**
sio_putch	Write a character into the driver's output buffer.
sio_write	Write a block of data to the driver's output buffer.
sio_SetWriteTimeouts	Set timeout value for **sio_write()** and **sio_putch()**.
sio_GetWriteTimeouts	Get timeout value for **sio_write()** and **sio_putch()**.
sio_AbortWrite	Abort when blocked from writing a block of data for **sio_write()** and **sio_putch()**.

Port Status Inquiry This category includes functions to query the communication status of the serial port.	
Function Name	**Description**
sio_lstatus	Get the status of the serial line.
sio_iqueue	Get the size of data accumulated in the system's input buffer and driver's input buffer.
sio_oqueue	Get the length of data not yet sent out in both the system's output buffer and the driver's output buffer.
sio_ofree	Get the length of free space in the driver's output buffer.
sio_Tx_hold	Check why data could not be transmitted.
sio_getbaud	Get the serial port's baud rate setting.
sio_getmode	Get the serial port's mode settings.
sio_getflow	Get the serial port's hardware and software flow control settings.
sio_data_status	Check if an error occurred when receiving data.

Event Control This category includes functions to set the communication event service routines for the serial port.	
Function Name	**Description**
sio_term_irq	Set an event service routine for the case when the terminator character is received.
sio_cnt_irq	Set an event service routine for the case when a certain amount of data has been received.
sio_modem_irq	Set an event service routine for the case when the line status is changed.
sio_break_irq	Set an event service routine for the case when a BREAK signal is received.
sio_Tx_empty_irq	Set an event service routine for the case when the last character in the output buffer was sent.

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

BSD Socket API

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

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

Socket Control
This category includes functions to open sockets, and set and retrieve communication parameters.

Function Name	Description
accept	Accept a connection on a socket.
bind	Associate a local address with a socket.
closesocket	Close a socket.
connect	Establish a connection to a peer.
ioctlsocket	Control the mode of a socket.
listen	Enable a socket to listen for incoming connection.
setsockopt	Set a socket option.
shutdown	Disable sends and/or receives on a socket.
socket	Create a socket.
getsockopt	Retrieve a socket option.

Data Input/Output
This category includes functions to read/write data from the socket.

Function Name	Description
recv	Receive data from a socket.
recvfrom	Receive a datagram and store the source address.
select	Determine the status of one or more sockets, waiting if necessary.
send	Send data on a connected socket.
sendto	Send data to a specific destination.

Socket Status Inquiry
This category includes functions to query the communication status from the socket.

Function Name	Description
getpeername	Get the address of the peer to which a socket is connected.
getsockname	Get the local name for a socket.
gethostname	Return the standard host name for the local machine.
gethostbyname	Get host information corresponding to a hostname.

Miscellaneous
This category includes special socket functions.

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

Simplified Socket API

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

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

Socket Control This category includes functions to open TCP/UDP sockets, and set and retrieve communication parameters.	
Function Name	**Description**
tcp_open	Open a local TCP port.
tcp_close	Close a local TCP port.
tcp_connect	Connect to specific host IP and port.
tcp_listen	Places a socket in a state where it is listening for an incoming connection.
tcp_listento	Listen for a specific incoming connection.
tcp_connect_nowait	Connect to specific host IP and port without waiting.
tcp_listen_nowait	Places a socket in a state where it is listening for an incoming connection without waiting.
tcp_listento_nowait	Listen for a specific incoming connection without waiting.
udp_open	Open a local UDP port.
udp_close	Close a local UDP port.

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

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

Port Status Inquiry This category includes functions to query current ethernet port status and parameters.	
Function Name	**Description**
net_get_IP	Get local IP address.
net_get_netmask	Get local subnet mask.
net_get_gateway	Get local default gateway.
net_get_MAC_address	Get MAC address.

System Control API

This section presents the system information library routines, and gives a brief description of each routine. For a more detailed description of these routines, see Chapter 3.

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

Function Name	**Description**
sys_clock_s	Read the server's time (seconds) count from power-up.
sys_clock_ms	Read the server's time (milliseconds) count from power-up.
sys_sleep_ms	Sleep task for some time (milliseconds).
sys_timeout	Set the timeout event service routine.
sys_get_info	Get server general information.
sys_restart_system	Restart system.
sys_restart_UserAP	Restart user AP.
sys_Set_RegisterID	Set Application ID.
sys_get_SerialType	Get async port interface signal type.
sys_set_SerialType	Set async port interface signal type.
sys_exit	Exit application.
sysc_SetName	Set server name.
sysc_SetPassword	Set password.
sysc_SetIP	Set the specified network interface IP address.
sysc_SetNetmask	Set the specified network interface netmask.
sysc_SetGateway	Set NPort gateway address.
sysc_SetIPConfig	Define how to get the IP address, netmask and gateway.
sysc_GetIPLocating	Set NPort IP Location function.
sysc_SetDebug	Set debug output setting.
sysc_SetSerialIoctl	Set serial port parameter.
sysc_SetSerialFIFO	Set the serial port FIFO settings.
sysc_GetSerialInterface	Set the serial port interface.
sysc_SaveAndRestart	Save the new settings and restart the NPort.
sysc_GetName	Get server name.
sysc_GetPassword	Get server password.
sysc_GetIP	Get the specified network interface IP address.
sysc_GetNetmask	Get the specified network interface netmask.
sysc_GetGateway	Get the specified network interface gateway.
sysc_GetIPConfig	Get the IP configuration settings.

Function Name	Description
sysc_GetIPLocating	Get NPort IP Location setting.
sysc_GetDebug	Get debug output setting.
sysc_GetSerialIoctl	Get serial port parameter.
sysc_GetSerialFIFO	Get the serial port FIFO settings.
sysc_GetSerialInterface	Get the serial port interface.
sysc_SetToDefault	Set to default value.
sys_GetServersIp	Retrieve the DNS servers' and time server's addresses supplied by DHCP/BOOTP server.
sys_LED	LED control API function.
sys_Buzzer	Buzzer control API function.
sys_GetRTC	Get Real Time Clock value.
sys_SetRTC	Set Real Time Clock value.
sys_get_LastErrno	Get last error number related to a socket.

Flash ROM Access API

This section presents the Flash Library routines, and gives a brief description of each routine. For a more detailed description of these routines, see Chapter 3.

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

Function Name	Description
flash_erase	Erase flash-ROM.
flash_length	Get current data length at the flash-ROM.
flash_write	Write data to the flash-ROM.
flash_read	Read data from the flash-ROM.
sys_FlashErase	Erase flash ROM.
sys_FlashLength	Get current data length at the flush-ROM.
sys_FlashWrite	Write data to flush-ROM.
sys_FlashRead	Read data to flush-ROM.

Debug API

This section presents the Debug Library routines, and gives a brief description of each routine. For a more detailed description of these routines, see Chapter 3.

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

Function Name	Description
dbg_put_ch	Print out a character for debugging.
dbg_put_block	Print out a block of data for debugging.
dbg_put_word	Print out a 2-byte unsigned integer value for debugging.
dbg_put_doubleword	Print out a 4-byte unsigned long value for debugging.
dbg_put_word_hex	Print out a 2-byte unsigned integer value with HEX format for debugging.

Function Name	Description
dbg_put_doubleword_hex	Print out a 4-byte unsigned long value with HEX format for debugging.
dbg_put_IP	Print out an IP address in the a.b.c.d format for debugging.
dbg_put_string	Print out a string for debugging.
dbg_printf	Print formatted output to the debug output stream.

Thread Control API

This section presents the Thread Control Library routines, and gives a brief description of each routine. For a more detailed description of these routines, see Chapter 3.

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

Function Name	Description
sys_ThreadCreate	Create a thread.
sys_ThreadClose	Close a thread.
sys_ThreadSuspend	Suspend a thread.
sys_ThreadResume	Resume a thread.
sys_ThreadState	Get a thread state.

Time Server API

This section presents the Time Server Library routines, and gives a brief description of each routine. For a more detailed description of these routines, see Chapter 3.

You should also note that the header files **sdkconf.h** and **sdksys.h** must be included in your source code when calling these functions.

Function Name	Description
sysc_getTimeServer	Return current time server used to syncronize the system time.
sysc_setTimeServer	Set time server that is used to syncronize the system time.
sysc_getTimeZone	Get the time offset used by time syncronization.
sysc_setTimeZone	Set the time offset used by time syncronization.
sysc_getTZoneIndex	Get the time zone of local system.
sysc_setTZoneIndex	Set the time zone of local system.
sys_GetLocalTime	Get local time.
sys_SetLocalTime	Set local time.

Memory Management API

This section presents the Memory Management Library routines, and gives a brief description of each routine. For a more detailed description of these routines, see Chapter 3.

You should also note that the header files **sdksys.h** must be included in your source code when calling these functions.

Function Name	Description
sys_malloc	Allocates memory blocks.
sys_calloc	Allocates an array in memory with elements initialized to 0.
sys_realloc	Reallocate memory blocks.
sys_free	Deallocates or frees a memory block.
sys_getFreeMemSize	Get available memory status used by **sys_malloc()** and **sys_calloc()**.

3

SDK API Reference

The following topics are covered in this chapter:

- **Serial I/O Library Reference**
- **BSD Socket Library Reference**
- **Simplified Socket Library Reference**
- **System Control Library Reference**
- **Flash ROM Access Library Reference**
- **Debug Library Reference**
- **Thread Control Library Reference**
- **Time Server Library Reference**
- **Memory Management Library Reference**

Serial I/O Library Reference

sio_AbortRead	Abort when blocked from reading a block of data for **sio_read()** and **sio_getch()**.	**Data Input**
C Format		
Syntax	**#include <sdksio.h>**	
	int sio_AbortRead (int *port*);	
Arguments	*port* Async serial port number.	
	1 NPort's serial port 1.	
	2 NPort's serial port 2.	
Description	Abort when blocked from reading a block of data for **sio_read()** and **sio_getch()**. Calling this function will cause **sio_read()** to return immediately with return code is the length of data read.	
Return Value	SIO_OK OK.	
	SIO_BADPORT Port number is invalid.	

sio_AbortWrite	Abort when blocked from writing a block of data for **sio_write()** and **sio_putch()**.	**Data Output**
C Format		
Syntax	**#include <sdksio.h>**	
	int sio_AbortWrite (int *port*);	
Arguments	*port* Async serial port number.	
	1 NPort's serial port 1.	
	2 NPort's serial port 2.	
Description	Abort when blocked from writing a block of data for **sio_write()** or **sio_putch()**. Calling this function will cause **sio_write()** to return immediately with return code is SIO_ABORT_WRITE.	
Return Value	SIO_OK OK.	
	SIO_BADPORT Port number is invalid.	

sio_ActXoff	This function causes transmission to act as if an XOFF character has been received.	**Misc.**
C Format		
Syntax	**#include <sdksio.h>**	
	int sio_ActXoff (int *port*);	
Arguments	*port* Async serial port number.	
	1 NPort's serial port 1.	
	2 NPort's serial port 2.	
Description	This function causes transmission to act as if an XOFF character has been received.	
Return Value	SIO_OK OK.	
	SIO_BADPORT Port number is invalid.	

sio_ActXon	This function causes transmission to act as if an XON character has been received.	**Misc.**
C Format		
Syntax	**#include <sdksio.h>**	
	int sio_ActXon (int *port*);	
Arguments	*port* Async serial port number.	
	1 NPort's serial port 1.	
	2 NPort's serial port 2.	
Description	This function causes transmission to act as if an XON character has been received.	
Return Value	SIO_OK OK.	
	SIO_BADPORT Port number is invalid.	

sio_baud	Set baud rate using the actual speed value.	**Port Control**
C Format		
Syntax	**#include <sdksio.h>**	
	int sio_baud (int *port*, long *speed*);	
Arguments	*port* Async serial port number.	
	1 NPort's serial port 1.	
	2 NPort's serial port 2.	
	speed True baud rate: e.g., 200, 1200, 9600, or 19200.	
Description	Set baud rate using the actual speed value.	
Return Value	SIO_OK OK	
	SIO_BADPORT Port number is invalid.	

sio_break	Send out a BREAK signal.	**Misc.**
C Format		
Syntax	**#include <sdksio.h>**	
	int sio_break (int *port*, int *time*);	
Arguments	*port* Async serial port number.	
	1 NPort's serial port 1.	
	2 NPort's serial port 2.	
	time Break time in tics (1/18.2 second).	
Description	This function will block transmission until the time has expired.	
Return Value	SIO_OK OK.	
	SIO_BADPORT Port number is invalid.	
	SIO_BADPARM Bad parameter.	
	SIO_NOT_OPEN Port was not open in advance.	

sio_break_ex	Send out a BREAK signal.	**Misc.**
C Format		
Syntax	**#include <sdksio.h>**	
	int sio_break_ex (int *port*, int *ms*);	
Arguments	*port* Async serial port number.	
	1 NPort's serial port 1.	
	2 NPort's serial port 2.	
	ms Break time in milliseconds.	
Description	Sends out a break signal. This function will block transmission until the time has expired, and is the same as **sio_break()**, except that the time unit is measured in milliseconds.	
Return Value	SIO_OK	OK.
	SIO_BADPORT	Port number is invalid.
	SIO_BADPARM	Bad parameter.
	SIO_NOT_OPEN	Port was not open in advance.

sio_break_irq	Set an event service routine for the case when a BREAK signal is received.	**Event Control**
C Format		
Syntax	**#include <sdksio.h>**	
	int sio_break_irq (int *port*, void (func*) (int *port*));**	
Arguments	*port* Async serial port number.	
	1 NPort's serial port 1.	
	2 NPort's serial port 2.	
	func Event service routine entry.	
	If func is NULL, this routine will be disabled.	
Description	Set an event service routine for the case when a BREAK signal is received. When a BREAK signal is encountered, the system will call the event service routine.	
Return Value	SIO_OK	OK.
	SIO_BADPORT	Port number is invalid.
	SIO_NOT_OPEN	Port was not open in advance.

sio_close	Disable a serial port for data transmitting/receiving.	**Port Control**
C Format		
Syntax	**#include <sdksio.h>**	
	int sio_close (int *port*);	
Arguments	*port* Async serial port number.	
	1 NPort's serial port 1.	
	2 NPort's serial port 2.	
Description	Disable a serial port so that it cannot receive/transmit data.	
Return Value	SIO_OK	OK
	SIO_BADPORT	Port number is invalid.
	SIO_NOT_OPEN	Port was not open in advance.

sio_cnt_irq	Set an event service routine for the case when a certain amount of data has been received.	**Event Control**
C Format		
Syntax	**#include <sdksio.h>**	
	int sio_cnt_irq (int *port*, void (func*) (int *port*), int *count*);**	
Arguments	*port* Async serial port number.	
	1 NPort's serial port 1.	
	2 NPort's serial port 2.	
	func Event service routine entry.	
	If *func* is NULL, this routine will be disabled.	
	count Data count.	
Description	Set an event service routine for the case when a certain amount of data has been received. When there are *count* bytes of data received in the input buffer, the system will call the event service routine.	
Return Value	SIO_OK	OK.
	SIO_BADPORT	Port number is invalid.
	SIO_NOT_OPEN	Port was not open in advance.

sio_data_status	Check if an error occurred when receiving data.	**Port Status**
C Format		
Syntax	**#include <sdksio.h>**	
	int sio_data_status (int *port*);	
Arguments	*port* Async serial port number.	
	1 NPort's serial port 1.	
	2 NPort's serial port 2.	
Description	Check if an error occurred when receiving data.	
Return Value	= 0	No error occurred.
	> 0	Bit 0 on - parity error.
		Bit 1 on - framing error.
		Bit 2 on - overrun error.
		Bit 3 on - overflow error.
	SIO_BADPORT	Port number is invalid.

sio_DTR	Set the DTR state of a port.	**Port Control**
C Format		
Syntax	**#include <sdksio.h>**	
	int sio_DTR (int *port,* int *mode*);	
Arguments	*port* Async serial port number.	
	1 NPort's serial port 1.	
	2 NPort's serial port 2.	
	mode 0 Turn DTR off.	
	1 Turn DTR on.	
Description	Set the DTR state of a port.	
Return Value	SIO_OK	OK
	SIO_BADPORT	Port number is invalid.
	SIO_BADPARM	Bad parameter
	SIO_NOT_OPEN	Port was not open in advance.

sio_flowctrl	Set hardware and/or software flow control. **Port Control**
C Format	
Syntax	**#include <sdksio.h>**
	int sio_flowctrl (int *port*, int *mode*);
Arguments	*port* Async serial port number.
	1 NPort's serial port 1.
	1 NPort's serial port 2.
	mode bit 0: CTS flow control
	bit 1: RTS flow control
	bit 2: Tx XON/XOFF flow control
	bit 3: Rx XON/XOFF flow control (0 = OFF, 1 = ON)
Description	Set the hardware and/or software flow control.
Return Value	SIO_OK OK
	SIO_BADPORT Port number is invalid.
	SIO_BADPARM Bad parameter
	SIO_NOT_OPEN Port was not open in advance.

sio_flush	Flush the driver's input/output buffer. **Port Control**
C Format	
Syntax	**#include <sdksio.h>**
	int sio_flush (int *port*, int *func*);
Arguments	*port* Async serial port number
	1 NPort's serial port 1
	2 NPort's serial port 2
	func flush action
	0 Flush input buffer.
	1 Flush output buffer.
	2 Flush input & output buffer.
Description	Flush the driver's input/output buffer. The data will no longer exist.
Return Value	SIO_OK OK
	SIO_BADPORT Port number is invalid.
	SIO_BADPARM Bad parameter

sio_getbaud	Get the serial port's baud rate setting.	**Port Status**
C Format		
Syntax	**#include <sdksio.h>**	
	long sio_getbaud (int *port*);	
Arguments	*port* Async serial port number.	
	1 NPort's serial port 1.	
	2 NPort's serial port 2.	
Description	Get the serial port's baud rate setting. The return value is the actual baud rate. For example, a return value of 9600 means 9600 bps whereas 200 means 200 bps.	
Return Value	> 0	The actual baud rate.
	SIO_BADPORT	Port number is invalid.

sio_getch	Read one character from the driver's input buffer.	**Data Input**
C Format		
Syntax	**#include <sdksio.h>**	
	int sio_getch (int *port*);	
Arguments	*port* Async serial port number.	
	1 NPort's serial port 1.	
	2 NPort's serial port 2.	
Description	Read one character from the driver's input buffer.	
Return Value	0 to 255	The ASCII code of the character received.
	SIO_BADPORT	Port number is invalid.
	SIO_NODATA	No data to read.
	SIO_BADPARM	Bad parameter.
	SIO_NOT_OPEN	Port was not open in advance.

sio_getflow	Get the serial port's hardware and software flow control settings.	**Port Status**
C Format		
Syntax	**#include <sdksio.h>**	
	int sio_getflow (int *port*);	
Arguments	*port* Async serial port number.	
	1 NPort's serial port 1.	
	2 NPort's serial port 2.	
Description	Get the serial port's hardware and software flow control settings. Refer to **sio_flowctrl()** for detail.	
Return Value	>=0 bit 0 =	CTS flow control.
	bit 1 =	RTS flow control.
	bit 2 =	Tx XON/XOFF flow control.
	bit 3 =	Rx XON/XOFF flow control.
	SIO_BADPORT	Port number is invalid.

sio_getmode	Get the serial port's mode settings. **Port Status**
C Format	
Syntax	**#include <sdksio.h>**
	int sio_getmode (int *port*);
Arguments	*port* Async serial port number.
	1 NPort's serial port 1.
	2 NPort's serial port 2.
Description	Get the serial port's mode settings. Refer to the description of **sio_ioctl()** to see the mode settings.
Return Value	>=0 The mode setting, see **sio_ioctl()** for detail.
	SIO_BADPORT Port number is invalid.

sio_GetReadTimeouts	Get timeout values for **sio_read()** and **sio_getch()**. **Data Input**
C Format	
Syntax	**#include <sdksio.h>**
	int sio_GetReadTimeouts (int *port*, DWORD **TotalTimeouts*, DWORD **IntervalTimeouts*);
Arguments	*port* Async serial port number.
	1 NPort's serial port 1.
	2 NPort's serial port 2.
	TotalTimeouts A pointer to buffer to retrieve total timeout value.
	IntervalTimeouts A pointer to buffer to retrieve interval timeout value.
Description	Get timeout values for **sio_read()** and **sio_getch()**.
Return Value	SIO_OK OK.
	SIO_BADPORT Port number is invalid.

sio_GetWriteTimeouts	Get timeout value for **sio_write()** and **sio_putch()**. **Data Output**
C Format	
Syntax	**#include <sdksio.h>**
	int sio_GetWriteTimeouts (int *port*, DWORD **TotalTimeouts*);
Arguments	*port* Async serial port number.
	1 NPort's serial port 1.
	2 NPort's serial port 2.
	TotalTimeouts A pointer to buffer to retrieve the total timeout value.
Description	Get timeout value for **sio_write()** and **sio_putch()**.
Return Value	SIO_OK OK.
	SIO_BADPORT Port number is invalid.

sio_ioctl	Control the settings of the serial port's I/O control register. **Port Control**
C Format	
Syntax	**#include <sdksio.h>** **int sio_ioctl (int *port*, int *baud*, int *mode*);**
Arguments	*port* Async serial port number. 1 NPort's serial port 1. 2 NPort's serial port 2. *baud* (bits/sec) 0 50bps 6 600bps 12 9600bps 1 75bps 7 1200bps 13 19200bps 2 110bps 8 1800bps 14 38400bps 3 134.5bps 9 2400bps 15 57600bps 4 150bps 10 4800bps 16 115200bps 5 300bps 11 7200bps 17 230400bps *mode* bit_cnt OR stop_bit OR parity. bit_cnt (bits 0-1) = 0x00 Data bit = 5 0x01 Data bit = 6 0x02 Data bit = 7 0x03 Data bit = 8 stop_bit (bit 2) = 0x00 Stop bit = 1 0x04 Stop bits = 1.5 or 2 parity (bits 3-5) = 0x00 No parity 0x08 Odd parity 0x18 Even parity 0x28 Mark parity 0x38 Space parity
Description	Control the settings of the serial port's I/O control register, such as baud rate, parity, data bits, and stop bit.
Return Value	SIO_OK OK SIO_BADPORT Port number is invalid. SIO_BADPARM Bad parameter SIO_NOT_OPEN Port was not open in advance.

sio_iqueue	Get the size of data accumulated in the system's input buffer and driver's input buffer.	**Port Status**
C Format		
Syntax	**#include <sdksio.h>**	
	long sio_iqueue (int *port*);	
Arguments	*port* Async serial port number.	
	1 NPort's serial port 1.	
	2 NPort's serial port 2.	
Description	Get the size of data accumulated in the system's input buffer and driver's input buffer.	
	User must be aware of the fact that there may be a few characters still in the RS-232 UART chip and not yet known when **sio_iqueue()** returns a zero value.	
Return Value	>= 0	Data current in input buffer, in bytes.
	SIO_BADPORT	Port number is invalid.

sio_lctrl	Set both the DTR and RTS states.	**Port Control**
C Format		
Syntax	**#include <sdksio.h>**	
	int sio_lctrl (int *port*, int *mode*);	
Arguments	*port* Async serial port number.	
	1 NPort's serial port 1.	
	2 NPort's serial port 2.	
	mode C_DTR (bit 0) C_RTS (bit 1)	
Description	Set both the DTR and RTS states.	
Return Value	SIO_OK	OK
	SIO_BADPORT	Port number is invalid.
	SIO_BADPARM	Bad parameter
	SIO_NOT_OPEN	Port was not open in advance.
	SIO_RTS_BY_HW	Can't control the port because it is set as auto H/W flow control by **sio_flowctrl()**.

sio_linput	Read a block of data ending with termination character.	**Data Input**
C Format		
Syntax	**#include <sdksio.h>**	
	int sio_linput (int *port*, char **buf*, int *len*, int *term*);	
Arguments	*port* Async serial port number.	
	1 NPort's serial port 1.	
	2 NPort's serial port 2.	
	buf Receive buffer pointer.	
	len Buffer length, in bytes.	
	term Terminator code.	
Description	Read a block of data from the driver's input buffer until the terminator character is encountered or *len* bytes of data are read.	
Return Value	> 0	Length of data received, in bytes.
	= 0	No data received.
	SIO_BADPORT	Port number is invalid.
	SIO_BADPARM	Bad parameter.
	SIO_NOT_OPEN	Port was not open in advance.

sio_lstatus	Get the status of the serial line.	**Port Status**
C Format		
Syntax	**#include <sdksio.h>**	
	int sio_lstatus (int *port*);	
Arguments	*port* Async serial port number.	
	1 NPort's serial port 1.	
	2 NPort's serial port 2.	
Description	Get the status of the line.	
Return Value	>= 0	Current line status.
		Bit 0 – S_CTS
		Bit 1 – S_DSR
		Bit 2 – S_RI
		Bit 3 – S_CD
	SIO_BADPORT	Port number is invalid.

sio_modem_irq	Set an event service routine for the case when the line status is changed.	**Event Control**
C Format		
Syntax	**#include <sdksio.h>**	
	int sio_modem_irq (int *port*, void (func*) (int *port*));**	
Arguments	*port* Async serial port number.	
	1 NPort's serial port 1.	
	2 NPort's serial port 2.	
	func Event service routine entry.	
	If the *func* is NULL, it will disable this routine.	
Description	Set an event service routine for the case when the line status has changed. When line status (CTS, DSR, CD, RI) changes, the system will call the event service routine.	
Return Value	SIO_OK	OK.
	SIO_BADPORT	Port number is invalid.
	SIO_NOT_OPEN	Port was not open in advance.

sio_ofree	Get the length of free space in the driver's output buffer.	**Port Status**
C Format		
Syntax	**#include <sdksio.h>**	
	long sio_ofree (int *port*);	
Arguments	*port* Async serial port number.	
	1 NPort's serial port 1.	
	2 NPort's serial port 2.	
Description	Get the length of free space in the driver's output buffer.	
Return Value	>= 0	Free space in output buffer, in bytes.
	SIO_BADPORT	Port number is invalid.

sio_open	Enable a serial port for data transmitting/receiving.	**Port Control**
C Format		
Syntax	**#include <sdksio.h>**	
	int sio_open (int *port*);	
Arguments	*port* Async serial port number.	
	1 NPort's serial port 1.	
	2 NPort's serial port 2.	
Description	Enable a serial port for data transmitting/receiving. After calling **sio_open()**, the initial status of this serial port is the same as the last setting or configuration setting.	
Return Value	>= 0	Open action was successful, and this return value is a descriptor referencing the port. The programmer can use this descriptor in the **select()** function (from the socket API group) to carry out a data read/write operation.
	SIO_BADPORT	Port number is invalid.

sio_oqueue	Get the length of the data in the output buffers	**Port Status**
C Format		
Syntax	**#include <sdksio.h>**	
	long sio_oqueue (int *port*);	
Arguments	*port* Async serial port number.	
	1 NPort's serial port 1.	
	2 NPort's serial port 2.	
Description	Get the length of the data in the system's output buffer and the driver's output buffer that has not yet been transmitted	
Return Value	>= 0	Length of data, in bytes, still remains in the driver's output buffer.
	SIO_BADPORT	Port number is invalid.

sio_putch	Write a character into the driver's output buffer.	**Data Output**
C Format		
Syntax	**#include <sdksio.h>**	
	int sio_putch (int *port*, int *term*);	
Arguments	*port* Async serial port number.	
	1 NPort's serial port 1.	
	2 NPort's serial port 2.	
	term The character to be written.	
Description	Write a character into the driver's output buffer.	
Return Value	SIO_OK	OK.
	SIO_BADPORT	Port number is invalid.
	SIO_BADPARM	Bad parameter.
	SIO_ABORT_WRITE	User abort blocked write.
	SIO_WRITETIMEOUT	Write timeout has occurred.
	SIO_NOT_OPEN	Port was not open in advance.

sio_read	Read data from the driver's input buffer.	**Data Input**
C Format		
Syntax	**#include <sdksio.h>**	
	int sio_read (int *port*, char **buf*, int *len*);	
Arguments	*port* Async serial port number.	
	1 NPort's serial port 1.	
	2 NPort's serial port 2.	
	buf Receive buffer pointer.	
	len Buffer length, in bytes.	
Description	Read data from the driver's input buffer. If the length of data in the driver's input buffer is less than the user's buffer, then all data in the driver's input buffer will be transferred to the user's buffer. Otherwise, only 'len' bytes will be transferred to the user's buffer. **sio_SetReadTimeout()** can be used to set timeouts for **sio_read()**. **sio_AbortRead()** can be used to abort any blocked **sio_read()**.	
Return Value	> 0	Length of data received, in bytes.
	= 0	No data received.
	SIO_BADPORT	Port number is invalid.
	SIO_BADPARM	Bad parameter.
	SIO_NOT_OPEN	Port was not open in advance.

sio_RTS	Set the RTS state of a port. **Port Control**
C Format	
Syntax	**#include <sdksio.h>**
	int sio_RTS (int *port*, int *mode*);
Arguments	*port* Async serial port number.
	1 NPort's serial port 1.
	2 NPort's serial port 2.
	mode 0 Turn RTS off.
	1 Turn RTS on.
Description	Set the RTS state of a port.
Return Value	SIO_OK OK
	SIO_BADPORT Port number is invalid.
	SIO_BADPARM Bad parameter.
	SIO_RTS_BY_HW Can't control the port because it is set as auto H/W flow control by **sio_flowctrl()**.
	SIO_NOT_OPEN Port was not open in advance.

sio_SetReadTimeouts	Set timeout values for **sio_read()** and **sio_getch()**. **Data Input**
C Format	
Syntax	**#include <sdksio.h>**
	int sio_SetReadTimeouts (int *port*, DWORD *TotalTimeouts*, DWORD *IntervalTimeouts*);
Arguments	*port* Async serial port number.
	1 NPort's serial port 1.
	2 NPort's serial port 2.
	TotalTimeouts Total timeout values, in milliseconds.
	IntervalTimeouts Interval timeout values, in milliseconds.
Description	Set timeout values for **sio_read()** and **sio_getch()**. The default *TotalTimeouts* value is MAXDWORD where the *IntervalTimeouts* value is 0, which enables **sio_read()** to return immediately.
Return Value	SIO_OK OK.
	SIO_BADPORT Port number is invalid.

sio_SetWriteTimeouts	Set timeout value for **sio_write()** and **sio_putch()**. **Data Output**	
C Format		
Syntax	**#include <sdksio.h>**	
	int sio_SetWriteTimeouts (int *port*, DWORD *TotalTimeouts*);	
Arguments	*port*	Async serial port number. 1 NPort's serial port 1. 2 NPort's serial port 2.
	TotalTimeouts	Total timeout values, in milliseconds.
Description	Set timeout value for **sio_write()** and **sio_putch()**. The default value of write timeout is MAXDWORD, which enables **sio_write()** and **sio_putch()** to return immediately without blocking at all. The value 0 enables **sio_write()** to always block until finished writing data.	
Return Value	SIO_OK	OK.
	SIO_BADPORT	Port number is invalid.
	SIO_BADPARM	Bad parameter.

sio_term_irq	Set an event service routine for the case when the terminator character is received. **Event Control**	
C Format		
Syntax	**#include <sdksio.h>**	
	int sio_term_irq (int *port*, void (func*) (int *port*), char *code*);**	
Arguments	*port*	Async serial port number. 1 NPort's serial port 1. 2 NPort's serial port 2.
	func	Event service routine entry. If the *func* is NULL, it will disable this routine.
	code	Terminator character code.
Description	Set an event service routine for the case when the terminator character is received. When the terminator character is received, the system will call the event service routine.	
Return Value	SIO_OK	OK.
	SIO_BADPORT	Port number is invalid.
	SIO_NOT_OPEN	Port was not open in advance.

sio_Tx_empty_irq	Set an event service routine for the case when the last character in the output buffer was sent.	**Event Control**
C Format		
Syntax	**#include <sdksio.h>**	
	int sio_Tx_empty_irq (int *port*, void (func*) (int *port*));**	
Arguments	*port* Async serial port number.	
	1 NPort's serial port 1.	
	2 NPort's serial port 2.	
	func Event service routine entry	
	If the func is NULL, it will disable this routine.	
Description	Set an event service routine for the case when the last character in the output buffer was sent. When the Tx empty signal is encountered, the system will call the event service routine.	
Return Value	SIO_OK	OK.
	SIO_BADPORT	Port number is invalid.
	SIO_NOT_OPEN	Port was not open in advance.

sio_Tx_hold	Check why data could not be transmitted.	**Port Status**
C Format		
Syntax	**#include <sdksio.h>**	
	int sio_Tx_hold (int *port*);	
Arguments	*port* Async serial port number.	
	1 NPort's serial port 1.	
	2 NPort's serial port 2.	
Description	Check the reason why data could not be transmitted.	
Return Value	>=0	Bit 0 on if why data could not transmitted is because CTS is low.
		Bit 1 on if why data could not transmitted is because XOFF character is received.
	SIO_BADPORT	Port number is invalid.

sio_write	Write a block of data to the driver's output buffer. **Data Output**	
C Format		
Syntax	**#include <sdksio.h>**	
	int sio_write (int *port*, char **buf*, int *len*);	
Arguments	*port* Async serial port number.	
	1 NPort's serial port 1.	
	2 NPort's serial port 2.	
	buf Transmit string pointer.	
	len Transmit string length, in bytes.	
Description	Write a block of data to the driver's output buffer. The actual length of data written depends on the amount of free space in the driver's output buffer. **sio_write()** is always non-block by default.	
	Use **sio_SetWriteTimeout()** to set timeout for **sio_write()**. SIO_WRITETIMEOUT will be returned from **sio_write()** when write timeout is set and timeout has occurred..	
	sio_AboutWrite() can be used to abort any blocked **sio_write()** with return value SIO_ABORT_WRITE.	
Return Value	>= 0	Length of data transmitted, in bytes.
	SIO_BADPORT	Port number is invalid.
	SIO_BADPARM	Bad parameter.
	SIO_ABORT_WRITE	User abort blocked write.
	SIO_WRITETIMEOUT	Write timeout has occurred.
	SIO_NOT_OPEN	Port was not open in advance.

BSD Socket Library Reference

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

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

closesocket	Close a socket. **Socket Control**
C Format	
Syntax	**#include <sdksock.h>** **int closesocket (int *s*);**
Arguments	*s* A descriptor identifying a socket.
Description	This function closes a socket. More precisely, it releases the socket descriptor *s*, so that further references to *s* will fail with the error EBADF. If this is the last reference to the underlying socket, the associated naming information and queued data are discarded. The semantics of **closesocket()** are affected by the socket options SO_LINGER and SO_DONTLINGER as follows: Option / Interval / Type of close / Wait for close? SO_DONTLINGER / Don't care / Graceful / No SO_LINGER / Zero / Hard / No SO_LINGER / Non-zero / Graceful / Yes If SO_LINGER is set (i.e., the *l_onoff* field of the **linger** structure is non-zero) with a zero timeout interval (*l_linger* is zero), **closesocket()** is not blocked even if queued data has not yet been sent or acknowledged. This is called a "hard" or "abortive" close, because the socket's virtual circuit is reset immediately, and any unsent data is lost. If SO_LINGER is set with a non-zero timeout interval, the **closesocket()** call blocks until the remaining data has been sent or until the timeout expires. This is called a graceful disconnect. If SO_DONTLINGER is set on a stream socket (i.e. the *l_onoff* field of the **linger** structure is zero), the **closesocket()** call will return immediately. However, any data queued for transmission will be sent if possible before the underlying socket is closed. This is also called a graceful disconnect. Note that in this case the implementation may not release the socket and other resources for an arbitrary period, which may affect applications which expect to use all available sockets.
Return Value	If no error occurs, **closesocket()** returns 0. Otherwise, it returns -1, and the global variable *errno* contains one of the following values.
Error Codes	EBADF The descriptor is not a socket.

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

gethostbyname	Get host information corresponding to a hostname. **Inquiry**
Syntax	**#include <sdksock.h>** **struct hostent *gethostbyname (char ****name*** **);**
Arguments	*name* A pointer to the name of the host.

Description	**gethostbyname()** returns a pointer to the following structure which contains the name(s) and address which correspond to the given address.
	struct hostent {
	char * h_name;
	char ** h_aliases;
	short h_addrtype;
	short h_length;
	char ** h_addr_list;
	};
	The members of this structure are:
	Element Usage
	h_name server name of local system
	h_aliases A NULL-terminated array of alternate names, unused currently.
	h_addrtype The type of address being returned; this is always AF_INET.
	h_length The length, in bytes, this is always 4
	h_addr_list A NULL-terminated list of addresses for the host. Addresses are returned in network byte order.
	The pointer returned points to a structure that is allocated by NPort Server. The application must not modify this structure or free any of its components.

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

gethostname	Return the standard host name for the local machine.	**Inquiry**
C Format		
Syntax	**#include <sdksock.h>**	
	int gethostname (char *_name_, int _namelen_);	
Arguments	*name*	A pointer to a buffer that will receive the host name.
	namelen	The length of the buffer.
Description	This routine returns the name of the local host into the buffer specified by the *name* parameter. The host name is returned as a null-terminated string. The form of the host name is dependent on the sockets implementation—it is a simple host name. However, it is guaranteed that the name returned will be successfully parsed by **gethostbyname()**.	
Return Value	If no error occurs, **gethostname()** returns 0. Otherwise, it returns -1, and the global variable *errno* contains one of the following values.	
Error Codes	EFAULT	The *namelen* parameter is too small or the name argument pointer is invalid.

getpeername	Get the address of the peer to which a socket is connected.	**Inquiry**
C Format		
Syntax	**#include <sdksock.h>**	
	int getpeername (int *s*, SOCKADDR *_name_, int *_namelen_);	
Arguments	*s*	A descriptor identifying a connected socket.
	name	The structure which is to receive the name of the peer.
	namelen	A pointer to the size of the *name* structure.
Description	**getpeername()** retrieves the name of the peer connected to the socket *s* and stores it in the SOCKADDR identified by *name*. It is used on a connected datagram or stream socket. On return, the *namelen* argument contains the actual size of the name returned in bytes.	
Return Value	If no error occurs, **getpeername()** returns 0. Otherwise, it returns -1, and the global variable *errno* contains one of the following values.	
Error Codes	EFAULT	The *name* argument pointer is invalid or *namelen* argument is not large enough.
	ENOTCONN	The socket is not connected.
	EBADF	The descriptor is not a socket.

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

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

htonl	Convert an unsigned long from host to network byte order. **Misc.**
C Format	
Syntax	**#include <sdksock.h>**
	u_long htonl (u_long *hostlong*);
Arguments	*hostlong* A 32-bit number in host byte order.
Description	This routine takes a 32-bit number in host byte order and returns a 32-bit number in network byte order.
Return Value	**htonl()** returns the value in network byte order.

htons	Convert an unsigned short from host to network byte order. **Misc.**
C Format	
Syntax	**#include <sdksock.h>**
	u_short htons (u_short *hostshort*);
Arguments	*hostshort* A 16-bit number in host byte order.
Description	This routine takes a 16-bit number in host byte order and returns a 16-bit number in network byte order.
Return Value	**htons()** returns the value in network byte order.

inet_addr	Convert a string containing a dotted address into a long integer. **Misc.**
C Format	
Syntax	**#include <sdksock.h>** **unsigned long inet_addr (char *_cp_);**
Arguments	*cp* A character string representing a number expressed in the Internet standard ".'' notation.
Description	This function interprets the character string specified by the *cp* parameter. This string represents a numeric Internet address expressed in the Internet standard ".'' notation. The value returned is a number suitable for use as an Internet address. All Internet addresses are returned in network order (bytes ordered from left to right). Internet addresses specified using the ".'' notation take the following form: a.b.c.d When four parts are specified, each is interpreted as a byte of data and assigned, from left to right, to the four bytes of an Internet address. Note that when an Internet address is viewed as a 32-bit integer quantity on the Intel architecture, the bytes referred to above appear as "d.c.b.a''. That is, the bytes on an Intel processor are ordered from right to left.
Return Value	If no error occurs, **inet_addr()** returns an **unsigned long** containing a suitable binary representation of the Internet address given. If the passed-in string does not contain a legitimate Internet address, for example if a portion of an "a.b.c.d" address exceeds 255, **inet_addr()** returns the value INADDR_ANY.

inet_ntoa	Convert a network address into a string in dotted format. **Misc.**
C Format	
Syntax	**#include <sdksock.h>** **char *inet_ntoa (unsigned long *in*);**
Arguments	*in* An Internet host address.
Description	This function takes an Internet address specified by the *in* parameter. It returns an ASCII string representing the address in ".'' notation as "a.b.c.d''. Note that the string returned by **inet_ntoa()** resides in memory which is allocated by the sockets implementation. The application should not make any assumptions about the way in which the memory is allocated. The data is guaranteed to be valid until the next sockets API call, but no longer.
Return Value	If no error occurs, **inet_ntoa()** returns a character pointer to a static buffer containing the text address in standard "." notation. Otherwise, it returns NULL. The data should be copied before another sockets call is made.

ioctlsocket	Control the mode of a socket. **Socket Control**	
C Format		
Syntax	**#include <sdksock.h>**	
	int ioctlsocket (int *s*, long *cmd*, u_long **argp*);	
Arguments	*s*	A descriptor identifying a socket.
	cmd	The command to perform on the socket *s*.
	argp	A pointer to a parameter for *cmd*.
Description	This routine may be used on any socket in any state. It is used to get or retrieve operating parameters associated with the socket, independent of the protocol and communication subsystem. The following commands are supported:	
	Command	Semantics
	FIONBIO	Enable or disable non-blocking mode on the socket *s*. *argp* points to an unsigned long, which is non-zero if non-blocking mode is to be enabled and zero if it is to be disabled. When a socket is created, it operates in blocking mode (i.e., non-blocking mode is disabled).
Return Value	Upon successful completion, the **ioctlsocket()** returns 0. Otherwise, it returns -1, and the global variable *errno* contains one of the following values.	
Error Codes	EINVAL	*cmd* is not a valid command, or *argp* is not an acceptable parameter for *cmd*, or the command is not applicable to the type of socket supplied.
	EBADF	The descriptor *s* is not a socket.

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

ntohl	Convert an unsigned long from network to host byte order. **Misc.**
C Format	
Syntax	**#include <sdksock.h>** **u_long ntohl (u_long *netlong*);**
Arguments	*netlong* A 32-bit number in network byte order.
Description	This routine takes a 32-bit number in network byte order and returns a 32-bit number in host byte order.
Return Value	**ntohl()** returns the value in host byte order.

ntohs	Convert an unsigned short from network to host byte order. **Misc.**
C Format	
Syntax	**#include <sdksock.h>** **u_short ntohs (u_short *netshort*);**
Arguments	*netshort* A 16-bit number in network byte order.
Description	This routine takes a 16-bit number in network byte order and returns a 16-bit number in host byte order.
Return Value	**ntohs()** returns the value in host byte order.

recv	Receive data from a socket. **Data Input/Output**	
C Format		
Syntax	**#include <sdksock.h>**	
	int recv (int *s*, char **buf*, int *len*, int *flags*);	
Arguments	*s*	A descriptor identifying a connected socket.
	buf	A buffer for the incoming data.
	len	The size of buffer pointed by *buf*.
	flags	Specifies the way in which the call is made.
Description	This function is used on datagram or connected stream sockets specified by the *s* parameter and is used to read incoming data. For sockets of type SOCK_STREAM, as much information as is currently available up to the size of the buffer supplied is returned. For datagram sockets, data is extracted from the first enqueued datagram, up to the size of the buffer supplied. If the datagram is larger than the buffer supplied, the buffer is filled with the first part of the datagram, and the excess data is lost. If no incoming data is available at the socket, the **recv()** call waits for data to arrive unless the socket is non-blocking. In this case a value of -1 is returned with the error code set to EWOULDBLOCK. The **select()** calls may be used to determine when more data arrives. If the socket is of type SOCK_STREAM and the remote side has shut down the connection gracefully or the connection has been reset, a **recv()** will complete immediately with 0 bytes received. *flags* may be used to influence the behavior of the function invocation beyond the options specified for the associated socket. That is, the semantics of this function are determined by the socket options and the *flags* parameter. The latter is constructed by "or-ing" any of the following values:	
	Value	Meaning
	MSG_OOB	Read out-of-band data (SOCK_STREAM only).
Return Value	If no error occurs, **recv()** returns the number of bytes received. If the connection has been closed, it returns 0. Otherwise, it returns -1, and the global variable *errno* contains one of the following values.	
Error Codes	EBADF	The descriptor is not a socket.
	EFAULT	The *buf* argument pointer is invalid.
	EOPNOTSUPP	MSG_OOB was specified, but the socket is not of type SOCK_STREAM.
	ESHUTDOWN	The socket has been shutdown; it is not possible to **recv()** on a socket after **shutdown()** has been invoked with *how* set to 0 or 2.
	EWOULDBLOCK	The socket is marked as non-blocking and the receive operation would block.
	EIO	MSG_OOB was specified, but has not received out-of-band data.
	ELENZERO	The length argument is zero.

recvfrom	Receive a datagram and store the source address.	**Data Input/Output**
C Format		
Syntax	**#include <sdksock.h>**	
	int recvfrom (int *s*, char **buf*, int *len*, int *flags*, SOCKADDR **from*, int **fromlen*);	
Arguments	*s*	A descriptor identifying a bound socket.
	buf	A buffer for the incoming data.
	len	The length of *buf*.
	flags	Specifies the way in which the call is made.
	from	An optional pointer to a buffer which will hold the source address upon return.
	fromlen	An optional pointer to the size of the *from* buffer.
Description	This function is used to read incoming data on a (possibly connected) socket and capture the address from which the data was sent. For sockets of type SOCK_STREAM, as much information as is currently available up to the size of the buffer supplied is returned. The *from* and *fromlen* parameters are ignored for SOCK_STREAM sockets. For datagram sockets, data is extracted from the first enqueued datagram, up to the size of the buffer supplied. If the datagram is larger than the buffer supplied, the buffer is filled with the first part of the message, and the excess data is lost. If *from* is non-zero, and the socket is of type SOCK_DGRAM, the network address of the peer which sent the data is copied to the corresponding SOCKADDR. The value pointed to by *fromlen* is initialized to the size of this structure, and is modified on return to indicate the actual size of the address stored there. If no incoming data is available at the socket, the **recvfrom()** call waits for data to arrive unless the socket is non-blocking. In this case a value of -1 is returned with the error code set to EWOULDBLOCK. The **select()** calls may be used to determine when more data arrives. If the socket is of type SOCK_STREAM and the remote side has shut down the connection gracefully or the connection has been reset, a **recvfrom()** will complete immediately with 0 bytes received. *flags* may be used to influence the behavior of the function invocation beyond the options specified for the associated socket. That is, the semantics of this function are determined by the socket options and the *flags* parameter. The latter is constructed by “or-ing” any of the following values: Value Meaning MSG_OOB Read out-of-band data (SOCK_STREAM only).	
Return Value	If no error occurs, **recvfrom()** returns the number of bytes received. If the connection has been closed, it returns 0. Otherwise, it returns -1, and the global variable *errno* contains one of the following values.	
Error Codes	EBADF	The descriptor is not a socket.
	EFAULT	The *buf* argument pointer is invalid.

EOPNOTSUPP	MSG_OOB was specified, but the socket is not of type SOCK_STREAM.
ESHUTDOWN	The socket has been shut down; it is not possible to **recvfrom()** on a socket after **shutdown()** has been invoked with *how* set to 0 or 2.
EWOULDBLOCK	The socket is marked as non-blocking and the receive operation would block.
EIO	MSG_OOB was specified, but has not received out-of-band data.
ELENZERO	The *fromlen* argument is zero.

select	Determine the status of one or more sockets, waiting if necessary.	**Data Input/Output**
C Format		
Syntax	**#include <sdksock.h>** **int select (int *nfds*, fd_set **readfds*, fd_set **writefds*, fd_set **exceptfds*, struct timeval **timeout*);**	
Arguments	*nfds*	This argument indicates the number of sockets to be checked.
	readfds	An optional pointer to a set of sockets to be checked for readability.
	writefds	An optional pointer to a set of sockets to be checked for writeability
	exceptfds	An optional pointer to a set of sockets to be checked for errors.
	timeout	The maximum time for **select()** to wait, or NULL for blocking operation.
Description	This function is used to determine the status of one or more sockets. For each socket, the caller may request information on read, write or error status. The set of sockets for which a given status is requested is indicated by an **fd_set** structure. Upon return, the structure is updated to reflect the subset of these sockets which meet the specified condition, and **select()** returns the number of sockets meeting the conditions. A set of macros is provided for manipulating an **fd_set**. These macros are compatible with those used in the Berkeley software, but the underlying representation is completely different. The first *nfds* descriptors are checked in each set; i.e., the descriptors from 0 through *nfds*-1 are examined. Note that this value should not exceed the number of sockets system allows; we recommand you pass the maximum number within the sockets you want to check plus 1 as this argument. Three independent sets of descriptors are watched. Those listed in *readfds* will be watched to see if characters become available for reading, those in *writefds* will be watched to see if it is OK to immediately write on them, and those in *exceptfds* will be watched for exceptions. On exit, the sets are modified in place to indicate which descriptors actually changed status.		

	Any of *readfds*, *writefds*, or *exceptfds* may be given as NULL if no descriptors are of interest. Four macros are defined in the header file **sdksock.h** for manipulating the descriptor sets. The variable FD_SETSIZE determines the maximum number of descriptors in a set (the default value of FD_SETSIZE is 96). Internally, an **fd_set** is represented as an array of int's. The macros are: FD_CLR(*s*, **set*) Removes the descriptor *s* from *set*. FD_ISSET(*s*, **set*) Nonzero if *s* is a member of the *set*, or zero otherwise. FD_SET(*s*, **set*) Adds descriptor *s* to *set*. FD_ZERO(**set*) Initializes the *set* to the NULL set. The parameter *timeout* controls how long the **select()** may take to complete. If *timeout* is a null pointer, **select()** will block indefinitely until at least one descriptor meets the specified criteria. Otherwise, *timeout* points to a struct timeval which specifies the maximum time that **select()** should wait before returning. If the timeval is initialized to {0, 0}, **select()** will return immediately; this is used to "poll" the state of the selected sockets.
Return Value	**select()** returns the total number of descriptors which are ready and contained in the **fd_set** structures, 0 if the time limit has expired. Otherwise, it returns -1, and the global variable *errno* contains one of the following values.
Error Codes	EINVAL The *readfds*, *writefds* and *exceptfds* are all NULL. EBADF One of the descriptor sets contains an entry which is not a socket.

send	Send data on a connected socket. **Data Input/Output**	
C Format		
Syntax	**#include <sdksock.h>**	
	int send (int *s*, const char **buf*, int *len*, int *flags*);	
Arguments	*s* A descriptor identifying a connected socket.	
	buf A buffer containing the data to be transmitted.	
	len The length of the data in *buf*.	
	flags Specifies the way in which the call is made	
Description	**send()** is used on connected datagram or stream sockets and is used to write outgoing data on a socket. For datagram sockets, care must be taken not to exceed the maximum IP packet size of the underlying subnets.	
	Note that the successful completion of a **send()** does not indicate that the data was successfully delivered.	
	If no buffer space is available within the transport system to hold the data to be transmitted, **send()** will block unless the socket has been placed in a non-blocking I/O mode. On non-blocking SOCK_STREAM sockets, the number of bytes written may be between 1 and the requested length, depending on buffer availability on both the local and foreign hosts. The **select()** call may be used to determine when it is possible to send more data.	
	Flags may be used to influence the behavior of the function invocation beyond the options specified for the associated socket. That is, the semantics of this function are determined by the socket options and the flags parameter. The latter is constructed by "or-ing" any of the following values:	
	Value Meaning	
	MSG_OOB Send out-of-band data.	
Return Value	If no error occurs, **send()** returns the total number of characters sent (note that this may be less than the number indicated by *len*.). Otherwise, it returns -1, and the global variable *errno* contains one of the following values.	
Error Codes	EFAULT	The *buf* argument is not in a valid part of the user address space.
	ENOTCONN	The socket is not connected.
	EBADF	The descriptor is not a socket.
	EOPNOTSUPP	MSG_OOB was specified, but the socket is not of type SOCK_STREAM.
	ESHUTDOWN	The socket has been shut down; it is not possible to **send()** on a socket after **shutdown()** has been invoked with *how* set to 1 or 2.
	EWOULDBLOCK	The socket is marked as non-blocking and the requested operation would block.
	EFBIG	Data written exceeds system capacity.

sendto	Send data to a specific destination. **Data Input/Output**
C Format	
Syntax	**#include <sdksock.h>** **int sendto (int *s*, char **buf*, int *len*, int *flags*, SOCKADDR **to*, int *tolen*);**
Arguments	*s* A descriptor identifying a socket *buf* A buffer containing the data to be transmitted. *len* The length of the data in *buf*. *flags* Specifies the way in which the call is made. *to* An optional pointer to the address of the target socket *tolen* The size of the address in *to*.
Description	**sendto()** is used on datagram or stream sockets and is used to write outgoing data on a socket. Note that the successful completion of a **sendto()** does not indicate that the data was successfully delivered. **sendto()** is normally used on a SOCK_DGRAM socket to send a datagram to a specific peer socket identified by the *to* parameter. On a SOCK_STREAM socket, the *to* and *tolen* parameters are ignored; in this case the **sendto()** is equivalent to **send()**. To send a broadcast (on a SOCK_DGRAM only), the address in the *to* parameter should be constructed using the special IP address INADDR_BROADCAST (defined in **sdksock.h**) together with the intended port number. It is generally inadvisable for a broadcast datagram to exceed the size at which fragmentation may occur, which implies that the data portion of the datagram (excluding headers) should not exceed 512 bytes. If no buffer space is available within the transport system to hold the data to be transmitted, **sendto()** will block unless the socket has been placed in a non-blocking I/O mode. On non-blocking SOCK_STREAM sockets, the number of bytes written may be between 1 and the requested length, depending on buffer availability on both the local and foreign hosts. The **select()** call may be used to determine when it is possible to send more data. *flags* may be used to influence the behavior of the function invocation beyond the options specified for the associated socket. That is, the semantics of this function is determined by the socket options and the flags parameter. The latter is constructed by "or-ing" any of the following values: Value Meaning MSG_OOB Send out-of-band data (SOCK_STREAM only).
Return Value	If no error occurs, **sendto()** returns the total number of characters sent (note that this may be less than the number indicated by *len*). Otherwise, it returns -1, and the global variable *errno* contains one of the following values.

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

setsockopt	Set a socket option.	**Socket Control**
C Format		
Syntax	**#include <sdksock.h>**	
	int setsockopt (int *s*, int *level*, int *optname*, char **optval*, int *optlen*);	
Arguments	*s*	A descriptor identifying a socket.
	level	The level at which the option is defined; the only supported level is SOL_SOCKET.
	optname	The socket option for which the value is to be set.
	optval	A pointer to the buffer in which the value for the requested option is supplied.
	optlen	The size of the *optval* buffer.
Description	**setsockopt()** sets the current value for a socket option associated with a socket of any type, in any state. Although options may exist at multiple protocol levels, this specification only defines options that exist at the uppermost “socket’’ level. Options affect socket operations, such as whether keep-connection message is sent in the normal data stream, whether **closesocket()** operation is graceful, etc. There are two types of socket options: Boolean options that enable or disable a feature or behavior, and options which require an integer value or structure. To enable a Boolean option, *optval* points to a nonzero integer. To disable the option, *optval* points to an integer equal to zero. *optlen* should be equal to sizeof(int) for Boolean options. For other options, *optval* points to the a structure that contains the desired value for the option, and *optlen* is the length of the structure.	

Description (cont.)

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

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

To enable SO_LINGER, the application should set *l_onoff* to a non-zero value, set *l_linger* to 0 or the desired timeout, in seconds, and call **setsockopt()**. The timeout value should be in the interval between 0 to10. To enable SO_DONTLINGER (i.e., disable SO_LINGER) *l_onoff* should be set to zero and *setsockopt()* should be called.

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

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

Value	Type	Meaning
SO_DONTLINGER	BOOL	Don't block close waiting for unsent data to be sent. Setting this option is equivalent to setting SO_LINGER with *l_onoff* set to zero.
SO_KEEPALIVE	BOOL	Send keepalives
SO_LINGER	LINGER	Linger on close if unsent data is present

Return Value

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

Error Codes

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

shutdown	Disable sends and/or receives on a socket. **Socket Control**	
C Format		
Syntax	**#include <sdksock.h>**	
	int shutdown (int *s*, int *how*);	
Arguments	*s*	A descriptor identifying a socket.
	how	A flag that describes what types of operation will no longer be allowed.
Description	**shutdown()** is used on all types of sockets to disable reception, transmission, or both. If *how* is 0, subsequent receives on the socket will be disallowed. This has no effect on the lower protocol layers. For TCP, the TCP window is not changed and incoming data will be accepted (but not acknowledged) until the window is exhausted. For UDP, incoming datagrams are accepted and queued. If *how* is 1, subsequent sends are disallowed. For TCP sockets, a FIN will be sent. Setting *how* to 2 disables both sends and receives as described above. Note that **shutdown()** does not close the socket, and resources attached to the socket will not be freed until **closesocket()** is invoked.	
Return Value	If no error occurs, **shutdown()** returns 0. Otherwise, it returns -1, and the global variable *errno* contains one of the following values.	
Error Codes	EINVAL	*how* is not valid
	ENOTCONN	The socket is not connected (SOCK_STREAM only).
	EBADF	The descriptor is not a socket

socket	Create a socket. **Socket Control**	
C Format		
Syntax	**#include <sdksock.h>**	
	int socket (int *af*, int *type*, int *protocol*);	
Arguments	*af*	An address format specification. The only format currently supported is AF_INET, which is the ARPA Internet address format.
	type	A type specification for the new socket.
	protocol	A particular protocol to be used with the socket, or 0 if the caller does not wish to specify a protocol.
Description	**socket()** allocates a socket descriptor of the specified address family, data type and protocol, as well as related resources. If a protocol is not specified (i.e., equal to 0), the default for the specified connection mode is used. Only a single protocol exists to support a particular socket type using a given address format. The protocol number to use is particular to the "communication domain" in which communication is to take place. The following type specifications are supported: SOCK_STREAM — Provides sequenced, reliable, two-way, connection-based byte streams with an out-of-band data transmission mechanism. Uses TCP for the Internet address family. SOCK_DGRAM — Supports datagrams, which are connectionless, unreliable buffers of a fixed (typically small) maximum length. Uses UDP for the Internet address family. Sockets of type SOCK_STREAM are full-duplex byte streams. A stream socket must be in a connected state before any data may be sent or received on it. A connection to another socket is created with a **connect()** call. Once connected, data may be transferred using **send()** and **recv()** calls. When a session has been completed, a **closesocket()** must be performed. The communications protocols used to implement a SOCK_STREAM to ensure that data is not lost or duplicated. SOCK_DGRAM sockets allow sending and receiving of datagrams to and from arbitrary peers using **sendto()** and **recvfrom()**. If such a socket is **connect()**'ed to a specific peer, datagrams may be sent to that peer using **send()** and may be received from (only) this peer using **recv()**.	
Return Value	If no error occurs, **socket()** returns a descriptor referencing the new socket. Otherwise, it returns -1, and the global variable *errno* contains one of the following values.	
Error Codes	EMFILE	No more file descriptors are available.
	EPROTONOSUPPORT	The specified address family or protocol is not supported.

Simplified Socket Library Reference

net_get_gateway	Get local default gateway. **Port Inquiry**
C Format	
Syntax	**#include <sdksys.h>** **u_long net_get_gateway (void);**
Arguments	N/A
Description	Get local default gateway.
Return Value	Default gateway IP address.

net_get_IP	Get local IP address. **Port Inquiry**
C Format	
Syntax	**#include <sdksys.h>** **u_long net_get_IP (void);**
Arguments	N/A
Description	Get local IP address.
Return Value	Local IP address.

net_get_MAC_address	Get MAC address. **Port Inquiry**
C Format	
Syntax	**#include <sdksys.h>** **void net_get_MAC_address (u_char *_mac_);**
Arguments	*mac* Get MAC address buffer pointer. This buffer must be 6-byte length.
Description	Get MAC address.
Return Value	System copies the host MAC address to the *mac* input buffer.

net_get_netmask	Get local subnet mask. **Port Inquiry**
C Format	
Syntax	**#include <sdksys.h>** **u_long net_get_netmask (void);**
Arguments	N/A
Description	Get local subnet mask.
Return Value	Local netmask.

tcp_close	Close a local TCP port. **Socket Control**
C Format	
Syntax	**#include <sdknet.h>** **int tcp_close (int *handle*);**
Arguments	*handle* The value returned from **tcp_open()**.
Description	Close a local TCP port.
Return Value	0 O.K. -1 Error handle number.

tcp_connect	Connect to specific host IP and port. **Socket Control**
C Format	
Syntax	**#include <sdknet.h>**
	int tcp_connect (int *handle*, u_long *rip*, int *rport*, long *tout*);
Arguments	*handle* The value return from **tcp_open()**.
	rip Remote host IP address that user wants to link.
	rport Remote host TCP port no.
	tout Wait for TCP connection time out value, in milliseconds. 0 will wait for OK or fail.
Description	Connect to specific host IP and port.
Return Value	1 Connect OK
	0 Connect fail.
	-1 Error handle number.
	-2 This handle is not a TCP handle.
	-3 Timeout counter reached.
	-4 Error state; already connected.
	-5 The *rip*:*rport* already in use.

tcp_connect_nowait	Connect to specific host IP and port without waiting. **Socket Control**
C Format	
Syntax	**#include <sdknet.h>**
	int tcp_connect_nowait (int *handle*, u_long *rip*, int *rport*);
Arguments	*handle* The value returned from **tcp_open()**.
	rip Remote host IP address that user wants to link to.
	rport Remote host's TCP port number.
Description	Connect to specific host's IP and port without waiting.
Return Value	0 Start to connect.
	-1 Error handle number.
	-2 Error argument.
	-3 Error state; already connected.
	-4 The *rip*:*rport* is already in use.

tcp_get_remote	Get connected host's IP and port. **Socket Inquiry**
C Format	
Syntax	**#include <sdknet.h>**
	int tcp_get_remote (int *handle*, u_long **rip*, int **rport*);
Arguments	*handle* The value returned from **tcp_open()**.
	rip Connected host's IP address pointer.
	rport Connected host's TCP port number pointer.
Description	Get connected host's IP and port.
Return Value	0 Get OK
	-1 Error handle.
	-2 Error argument.
	-3 No connection.

tcp_iqueue	Get the size of data accumulated in TCP driver's input buffer. **Socket Inquiry**
C Format	
Syntax	**#include <sdknet.h>**
	int tcp_iqueue (int *handle*);
Arguments	*handle* The value returned from **tcp_open()**.
Description	Get the size of data accumulated in TCP driver's input buffer.
Return Value	>=0 TCP input buffer queued data size.
	-1 Error handle number.
	-2 This is not a TCP handle.
	-3 TCP not connected.

tcp_listen	Places a socket in a state where it is listening for an incoming connection. **Socket Control**
C Format	
Syntax	**#include <sdknet.h>**
	int tcp_listen (int *handle*, long *tout*);
Arguments	*handle* The value return from **tcp_open()**.
	tout Wait for listen timeout value, in milliseconds.
	0 will wait for someone to connect.
Description	Places a socket a state where it is listening for an incoming connection.
Return Value	1 Connect OK or already connected.
	0 Connect fail.
	-1 Error handle number.
	-2 This handle is not a TCP handle.
	-3 Timeout counter reached.
	-4 Rrror state; already connected.

tcp_listen_nowait	Places a socket in a state where it is listening for an incoming connection without waiting.	**Socket Control**
C Format		
Syntax	**#include <sdknet.h>**	
	int tcp_listen_nowait (int *handle*);	
Arguments	*handle* The value returned from **tcp_open()**.	
Description	Places a socket a state where it is listening for an incoming connection without waiting.	
Return Value	0 Start to listen.	
	-1 Error handle number	
	-2 This handle is not a TCP handle.	
	-3 Error state; already connected.	

tcp_listento	Listen for a specific incoming connection.	**Socket Control**
C Format		
Syntax	**#include <sdknet.h>**	
	int tcp_listento (int *handle*, u_long *rip*, int *rport*, long *tout*);	
Arguments	*handle* The value returned from **tcp_open()**.	
	rip Temote host IP address that user wants to link to. 0 indicates don't care remote IP address.	
	rport Remote host TCP port number. 0 indicates don't care the TCP port number.	
	tout Wait for listen timeout value, in milliseconds.	
Description	Listen for a specific incoming connection.	
Return Value	1 Connect OK or already connected.	
	0 Connect fail.	
	-1 Error handle number.	
	-2 This handle is not a TCP handle.	
	-3 Timeout counter reached.	
	-4 Error state; already connected.	

tcp_listento_nowait	Listen for a specific incoming connection without waiting.	**Socket Control**
C Format		
Syntax	**#include <sdknet.h>**	
	int tcp_listento_nowait (int *handle*, u_long *rip*, int *rport*);	
Arguments	*handle*	The value returned from **tcp_open()**.
	rip	Remote host IP address that user wants to link to. 0 indicates don't care remote IP address.
	rport	Remote host's TCP port number. 0 indicates don't care the TCP port number.
Description	Listen for a specific incoming connection without waiting.	
Return Value	0	Start to listen.
	-1	Error handle number
	-2	This handle is not a TCP handle.
	-3	Error state; already connected.

tcp_ofree	Size of free space in TCP driver's input buffer.	**Socket Inquiry**
C Format		
Syntax	**#include <sdknet.h>**	
	int tcp_ofree (int *handle*);	
Arguments	*handle*	The value returned from **tcp_open()**.
Description	Size of free space in TCP driver's input buffer.	
Return Value	>=0	TCP output buffer's free size.
	-1	Error handle number.
	-2	This is not a TCP handle.
	-3	TCP not connected.

tcp_open	Open a local TCP port.	**Socket Control**
C Format		
Syntax	**#include <sdknet.h>**	
	int tcp_open (int *port*);	
Arguments	*port*	Local TCP port number.
Description	Open a local TCP port.	
Return Value	>=0	Open handle.
	-1	Open fail.

tcp_recv	Receives data from a connected socket. **Data Input/Output**
C Format	
Syntax	**#include <sdknet.h>**
	int tcp_recv (int *handle*, char **buffer*, int *len*);
Arguments	*handle* The value returned from **tcp_open()**.
	buffer The receiveed data buffer pointer.
	len The size of *buffer*, in bytes.
Description	Receives data from a connected socket.
Return Value	>=0 Received data length.
	-1 Error handle number.
	-2 Error argument.
	-3 TCP not connected.

tcp_send	Sends data on a connected socket. **Data Input/Output**
C Format	
Syntax	**#include <sdknet.h>**
	int tcp_send (int *handle*, char **buffer*, int *len*);
Arguments	*handle* The value return from **tcp_open()**.
	buffer The send out data buffer pointer.
	len The length of data in *buffer* to be sent, in bytes.
Description	Sends data on a connected socket.
Return Value	>=0 Send out data length.
	-1 Error handle number.
	-2 Error argument.
	-3 TCP not connected.

tcp_state	Get TCP state. **Socket Inquiry**
C Format	
Syntax	**#include <sdknet.h>**
	int tcp_state (int *handle*);
Arguments	*handle* The value returned from **tcp_open()**.
Description	Get TCP state.
Return Value	0 TCP closed.
	1 TCP listen.
	2 TCP connecting.
	3 TCP connected.
	4 TCP close wait (remote closed).
	5 TCP closing.
	-1 error handle.
	-2 This handle is not a TCP handle.

udp_close	Close a local UDP port. **Socket Control**
C Format	
Syntax	**#include <sdknet.h>** **int udp_close (int *handle*);**
Arguments	*handle* The value return from **udp_open()**.
Description	Close a local UDP port.
Return Value	0 Close OK. -1 Error handle number.

udp_iqueue	Get the size of data accumulated in UDP driver's input buffer. **Socket Inquiry**
C Format	
Syntax	**#include <sdknet.h>** **int udp_iqueue (int *handle*);**
Arguments	*handle* The value returned from **udp_open()**.
Description	Get the size of data accumulated in UDP driver's input buffer.
Return Value	>=0 UDP input buffer queued data size. -1 Error handle number. -2 This is not a UDP handle.

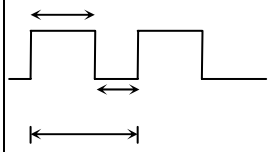
udp_ofree	Size of free space in UDP driver's output buffer. **Socket Inquiry**
C Format	
Syntax	**#include <sdknet.h>** **int udp_ofree (int *handle*);**
Arguments	*handle* The value returned from **udp_open()**.
Description	Size of free space in UDP driver's output buffer.
Return Value	>=0 UDP output buffer free size. -1 Error handle number. -2 This is not a UDP handle.

udp_open	Open a local UDP port. **Socket Control**
C Format	
Syntax	**#include <sdknet.h>** **int udp_open (int *port*);**
Arguments	*port* The local UDP port number.
Description	Open a local UDP port.
Return Value	>=0 Open handle. -1 Open fail.

udp_recv	Receives data from a specific source address. **Data Input/Output**
C Format	
Syntax	**#include <sdknet.h>** **int udp_recv (int *handle*, u_long **rip*, int **rport*, char **buf*, int *len*);**
Arguments	*handle* The value return from **udp_open()**. *rip* Return the remote host IP address of. *rport* Return the remote host UDP port number. *buf* Pointer to buffer to receive incoming data. *len* The length of *buf*, in bytes.
Description	Receives data from a specific source address.
Return Value	>= 0 Length of data received. -1 Receive failed.

udp_send	Sends data to a specific destination. **Data Input/Output**
C Format	
Syntax	**#include <sdknet.h>** **int udp_send (int *handle*, u_long *rip*, int *rport*, char **buf*, int *len*);**
Arguments	*handle* The value returned from **udp_open()**. *rip* Send to host IP address. *rport* Send to host UDP port number. *buf* Send data buffer pointer. *len* Send data length, in bytes.
Description	Sends data to a specific destination.
Return Value	>= 0 Sent out data length. -1 Send failed.

System Control Library Reference

sys_Buzzer	Buzzer control API function.
Syntax	**#include <sdksys.h>** **int sys_Buzzer (int *cnt*, int *ontime*, int *offtime*);**
Arguments	*Cnt* Number of buzzer on/off. *ontime* Buzzer on-time, in milliseconds. *offtime* Buzzer off-time, in milliseconds.
Description	This function control the buzzer as defined blow.  The buzzer function is for NPort 5210-P and NPort 5230-P only.
Return Value	-1 No support. 0 O.K.

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

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

sys_exit	Exit application.
C Format	
Syntax	**#include <sdksys.h>** **void sys_exit (*void*);**
Arguments	N/A
Description	Exit user application and return to kernel. It will stop the user application.
Return Value	N/A

sys_get_info	Get server general information.
C Format	
Syntax	**#include <sdksys.h>** **int sys_get_info (struct sdk_sysinfo *_info_);**
Arguments	*info* A pointer to a buffer that will receive the server general information.

Description	The returned information is with following structure:
	struct sdk_sysinfo {
	struct sdk_version firmware_version; /* Server's firmware version. */
	unsigned long serial_no; /* Server's serial number */
	unsigned short product_id; /* Server's product ID */
	unsigned char MAC_addr[6]; /* Server Ethernet MAC address */
	struct sdk_version ap_version; /* User's AP version */
	unsigned short ap_date_year; /* Date of AP: A.D. e.g. 2002 */
	unsigned char ap_date_month; /* Range: 1 - 12 */
	unsigned char ap_date_day; /* Range: 1 - 31 */
	unsigned char ap_time_hour; /* Range: 0 - 23 */
	unsigned char ap_time_minute; /* Range: 0 - 59 */
	};
	The version informations are stored as following structure:
	struct sdk_version {
	unsigned short ext_version;
	unsigned char sub_version;
	unsigned char main_version;
	};
	e.g., For version 1.20.3, the *main_version* is 1, *sub_version* is 20 and *ext_version* is 3.

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

sys_get_LastErrno	Get last error number related to a socket.
C Format	
Syntax	**#include <sdksys.h>**
	int sys_get_LastErrno (int *socket*, int **err*);
Arguments	*socket* A descriptor identifying a socket.
	err A pointer to where to place the last error number.
Description	Most sockets APIs will put the error reason as error number in a global variable *errno*. In some multi-threading application, the variable may be overwriten suddenly by another thread. In such case, you can retrieve the last error occurring to specified socket by using this function. On return, the error number is placed in the space specified by *err*.
	Note that the returned error number is meaningful only when the last socket operation on specified socket failed. Otherwise, the returned error number is undefined.
Return Value	-3 The pointer is invalid.
	0 O.K.

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

sys_GetRTC	Get Real Time Clock value.
Syntax	**#include <sdksys.h>** **void sys_GetRTC (struct rtc_time **timep*);**
Arguments	*timep* RTC time structure buffer pointer.

Description	The NE-4100 series use software timer to simulate real time clock. The NE-4100 use NTP (Network Time Protocol) to synchronize the date and time with time server. In case there's no time information required by NTP, the system time will be set to Jan.1, 1999.
	The RTC information returned in following structure.
	struct rtc_time {
	int t_sec; /* 00 - 59 */
	int t_min; /* 00 - 59 */
	int t_hour; /* 00 - 23 */
	int t_mday; /* 01 - 31 */
	int t_mon; /* 01 - 12 */
	int t_year; /* 2000 - 2099 */
	int t_wday; /* 01 - 07 */
	};

Return Value	-1 No support. 0 O.K.

sys_GetServersIp	Retrieve the DNS servers' and time server's addresses supplied by DHCP/BOOTP server.
C Format	
Syntax	**#include <sdksys.h>** **int sys_GetServersIp (unsigned long **dns1_ip*, unsigned long **dns2_ip*, unsigned long **time_ip*);**
Arguments	*dns1_ip* Pointer to buffer to retrieve the first DNS server's IP address. If *dns1_ip* is NULL, this address is not returned. *dns2_ip* Pointer to buffer to retrieve the second DNS server's IP address. If *dns2_ip* is NULL, this address is not returned. *time_ip* Pointer to buffer to retrieve time server's IP address. If *time_ip* is NULL, time server's IP is not returned.
Description	NPort's network configuration can be set to DHCP and/or BOOTP mode by **sysc_SetIPConfig()**. In such case, DHCP/BOOTP server may also supply the addresses of DNS servers and time server. **sys_GetServersIp()** returns these information to user program. If some information is not provided, a return value of 0 would be put in the buffer.
Return Value	0 O.K.

sys_LED	LED control API function.
C Format	
Syntax	**#include <sdksys.h>** **int sys_LED (int *LED1State*, int *LED2State*);**
Arguments	*LED1State* Green LED state. -1 Maintain current state. 0 Set LED off. 1 Set LED on. 2 Set LED shiny slow. 3 Set LED shiny fast. *LED2State* Red LED state. Same as above.
Description	LED control API function is just for NPort 5210-P, NPort 5230-P, NPort 5410-P, NPort 5430-P and NPort 5430I-P HW.
Return Value	-1 No support. -2 Error argument. 0 O.K.

sys_restart_system	Restart system.
C Format	
Syntax	**#include <sdksys.h>** **void sys_restart_system (void);**
Arguments	N/A
Description	Restart system.
Return Value	N/A

sys_restart_UserAP	Restart user AP.
C Format	
Syntax	**#include <sdksys.h>** **void sys_restart_UserAP (void);**
Arguments	N/A
Description	Restart user AP.
Return Value	N/A

sys_Set_RegisterID	Set Application ID.
C Format	
Syntax	**#include <sdksys.h>** **void sys_Set_RegisterID (u_long *id*);**
Arguments	*id* Application ID.
Description	It lets user the application ID. User just can use 0x00000000 to 0x7FFFFFFF and 0x80000000 to 0xFFFFFFFF is reversed for MOXA only. Your application should first call it after it starts to run.
Return Value	N/A

sys_set_SerialType	Set async port interface signal type.
C Format	
Syntax	**#include <sdksys.h>** **int sys_set_SerialType (int *port*, int *type*);**
Arguments	*port* Async serial port number. 1 NPort's serial port 1. 2 NPort's serial port 2. *type* 0 RS-232. 1 RS-422. 2 RS-485 2-wire. 3 RS-485 4-wire. *Note: Not supported by NPort 4511.*
Description	Set async port interface signal type.
Return Value	0 Set OK. -1 Bad port. -2 Bad parameter (cannot set this interface type).

sys_SetRTC	Set Real Time Clock value.
C Format	
Syntax	**#include <sdksys.h>** **void sys_SetRTC (struct rtc_time **timep*);**
Arguments	*timep* RTC time structure buffer pointer.

Description	Set the value of RTC. The RTC value is stored in following structure.
	struct rtc_time {
	int t_sec; /* 00 - 59 */
	int t_min; /* 00 - 59 */
	int t_hour; /* 00 - 23 */
	int t_mday; /* 01 - 31 */
	int t_mon; /* 01 - 12 */
	int t_year; /* 2000 - 2099 */
	int t_wday; /* 01 - 07 */
	};

Return Value	-1 No support. 0 O.K.

sys_sleep_ms	Sleep task for some time (milliseconds).
C Format	
Syntax	**#include <sdksys.h>** **int sys_sleep_ms (long *time_ms*);**
Arguments	*time_ms* Task sleep time in milliseconds.
Description	Task sleep time (milliseconds).
Return Value	This function always returns 0.

sys_timeout	Set the timeout event service routine.
C Format	
Syntax	**#include <sdksys.h>** **int sys_timeout (void (*_func_)(), long _time_ms_);**
Arguments	*func* The timeout event service routine. *time_ms* Timeout value in milliseconds.
Description	Set the timeout event service routine.
Return Value	0 No errors. EINVAL The isr argument event function pointer is invalid. ENOBUFS No resources.

sysc_GetDebug	Get debug output setting.
C Format	
Syntax	**#include <sdkconf.h>** **int sysc_GetDebug(void);**
Arguments	N/A
Description	Get debug output setting.
Return Value	0 Debug mode off. 1 Debug mode on.

sysc_GetGateway	Get the specified network interface gateway.
C Format	
Syntax	**#include <sdkconf.h>** **u_long sysc_GetGateway (void);**
Arguments	N/A
Description	Get server gateway.
Return Value	Gateway IP address.

sysc_GetIP	Get the specified network interface IP address.
C Format	
Syntax	**#include <sdkconf.h>** **u_long sysc_GetIP (void);**
Arguments	N/A
Description	Get server IP address.
Return Value	The local server IP address.

sysc_GetIPConfig	Get the IP configuration settings.
C Format	
Syntax	**#include <sdkconf.h>**
	int sysc_GetIPConfig (viod);
Arguments	N/A
Description	Get the IP configuration settings.
Return Value	0 Static IP.
	1 DHCP.
	2 DHCP & BOOTP.
	3 BOOTP.

sysc_GetIPLocating	Get NPort IP Location setting.
C Format	
Syntax	**#include <sdkconf.h>**
	int sysc_GetIPLocating (u_long **ipaddr*, u_int **pno*, int **time*);
Arguments	*ipaddr* Remote server ip address buffer pointer.
	pno Remote UDP port number pointer.
	time Report period time pointer.
Description	Get NPort IP Location setting.
Return Value	0 O.K.
	-2 Error argument.

sysc_GetName	Get server name.
Syntax	**#include <sdkconf.h>**
	int sysc_GetName (char **name*, int *size*);
Arguments	*name* Server name buffer pointer.
	size Buffer size.
Description	Get server name.
Return Value	>=0 The length of server name returned.

sysc_GetNetmask	Get the specified network interface netmask.
C Format	
Syntax	**#include <sdkconf.h>**
	u_long sysc_GetNetmask (void);
Arguments	N/A
Description	Get server netmask.
Return Value	The local server IP netmask.

sysc_GetPassword	Get server password.
C Format	
Syntax	**#include <sdkconf.h>**
	int sysc_GetPassword (char ****password*, int** ***size*** **);**
Arguments	*password* Server password buffer pointer.
	size Buffer size.
Description	Get server password.
Return Value	>=0 The length of password returned.

sysc_GetSerialFIFO	Get the serial port FIFO settings.
C Format	
Syntax	**#include <sdkconf.h>**
	int sysc_GetSerialFIFO (int ***port*** **);**
Arguments	*port* Async serial port number.
	1 NPort's serial port 1.
	2 NPort's serial port 2.
Description	Get the serial port FIFO settings.
Return Value	1 FIFO enabled.
	0 FIFO disabled.
	-1 Error port number.

sysc_GetSerialInterface	Get the serial port interface.
C Format	
Syntax	**#include <sdkconf.h>**
	int sysc_GetSerialInterface (int ***port*** **);**
Arguments	*port* Async serial port number.
	1 NPort's serial port 1.
	2 NPort's serial port 2.
Description	Get the serial port interface.
Return Value	-1 Error port number.
	0 RS-232.
	1 RS-422.
	2 RS-485 2-wire.
	3 RS-485 4-wire.

sysc_GetSerialIoctl	Get serial port parameter.
C Format	
Syntax	**#include <sdkconf.h>**
	int sysc_GetSerialIoctl (int *port*, int **baud*, int **mode*, int **flow*);
Arguments	*port* Async serial port number. 1 NPort's serial port 1. 2 NPort's serial port 2.
	baud Baud rate buffer pointer. 0 50bps 6 600bps 12 9600bps 1 75bps 7 1200bps 13 19200bps 2 110bps 8 1800bps 14 38400bps 3 134.5bps 9 2400bps 15 57600bps 4 150bps 10 4800bps 16 115200bps 5 300bps 11 7200bps 17 230400bps
	mode Character mode buffer pointer. bit_cnt (bits 0-1) = 0x00 Data bit=5. 0x01 Data bit=6. 0x02 Data bit=7. 0x03 Data bit=8. stop_bit (bit 2) = 0x00 Stop bit=1. 0x04 Stop bits=1.5 or 2. parity (bits 3-5) = 0x00 No parity. 0x08 Odd parity. 0x18 Even parity. 0x28 Mark parity. 0x38 Space parity.
	flow Flow control buffer pointer. 0 None. 1 RTS/CTS. 2 XON/XOFF. 3 DTR/DSR.
Description	Get serial port parameter.
Return Value	0 O.K. -1 Error port number.

sysc_SaveAndRestart	Save the new settings and restart the NPort.
C Format	
Syntax	**#include <sdkconf.h>** **void sysc_SaveAndRestart (void);**
Arguments	N/A
Description	Save the new settings and restart the NPort. After calling this function, NPort will be restared.
Return Value	N/A

sysc_SetDebug	Set debug output setting.
C Format	
Syntax	**#include <sdkconf.h>** **int sysc_SetDebug (int *mode*);**
Arguments	*mode* 0 Off. 1 On.
Description	Set debug output setting.
Return Value	0 O.K. -2 Error argument.

sysc_SetGateway	Set NPort gateway address.
C Format	
Syntax	**#include <sdkconf.h>** **int sysc_SetGateway (u_long *ipaddr*);**
Arguments	*ipaddr* New gateway IP address.
Description	Set gateway.
Return Value	0 O.K. -2 Error argument.

sysc_SetIP	Set the specified network interface IP address.
C Format	
Syntax	**#include <sdkconf.h>** **int sysc_SetIP (u_long *ipaddr*);**
Arguments	*ipaddr* New local server IP address.
Description	Set IP address.
Return Value	0 O.K. -2 Error argument.

sysc_SetIPConfig	Define how to get the IP address, netmask and gateway.
C Format	
Syntax	**#include <sdkconf.h>** **int sysc_SetIPConfig (int *type*);**
Arguments	*type* Get server IP address mode. 0 Static IP. 1 DHCP. 2 DHCP & BOOTP. 3 BOOTP.
Description	Define how to get the IP address, netmask and gateway.
Return Value	0 O.K. -2 Error argument.

sysc_SetIPLocating	Set NPort IP Location function.
C Format	
Syntax	**#include <sdkconf.h>** **int sysc_SetIPLocating (u_long *ipaddr*, u_int *pno*, int *time*);**
Arguments	*ipaddr* IP Location remote server IP address. Set 0.0.0.0 to disable this function. *pno* IP Location remote UDP port number. *time* report period time, in seconds.
Description	Set NPort IP Location function.
Return Value	0 O.K. -2 Error argument.

sysc_SetName	Set server name.
C Format	
Syntax	**#include <sdkconf.h>** **int sysc_SetName(char **name*);**
Arguments	*name* The new server name.
Description	Set server name.
Return Value	0 O.K.

sysc_SetNetmask	Set the specified network interface netmask.
C Format	
Syntax	**#include <sdkconf.h>** **int sysc_SetNetmask (u_long *netmask*);**
Arguments	*netmask* New local server netmask.
Description	Set netmask.
Return Value	0 O.K.

sysc_SetPassword	Set password.
C Format	
Syntax	**#include <sdkconf.h>**
	int sysc_SetPassword (char **password*);
Arguments	*password* The new server password.
Description	Set password.
Return Value	0 O.K.

sysc_SetSerialFIFO	Set the serial port FIFO settings.
C Format	
Syntax	**#include <sdkconf.h>**
	int sysc_SetSerialFIFO (int *port*, int *mode*);
Arguments	*port* Async serial port number.
	1 NPort's serial port 1.
	2 NPort's serial port 2.
	mode FIFO mode.
	0 Disable.
	1 Enable.
Description	Set the serial port FIFO settings.
Return Value	0 O.K.
	-1 Error port number.
	-2 Error argument.

sysc_SetSerialInterface	Set the serial port interface.
C Format	
Syntax	**#include <sdkconf.h>**
	int sysc_SetSerialInterface (int *port*, int *type*);
Arguments	*port* Async serial port number.
	1 NPort's serial port 1.
	2 NPort's serial port 2.
	type Serial interface type.
	0 RS-232.
	1 RS-422.
	2 RS-485 2-wire.
	3 RS-485 4-wire.
Description	Set the serial port interface.
Return Value	0 O.K.
	-1 Error port number.
	-2 Error argument.

sysc_SetSerialIoctl	Set serial port parameter.
C Format	
Syntax	**#include <sdkconf.h>** **int sysc_SetSerialIoctl (int *port*, int *baud*, int *mode*, int *flow*);**
Arguments	*port* Async serial port number. 1 NPort's serial port 1. 2 NPort's serial port 2. *baud* 0 50bps, 1 75bps, 2 110bps, 3 134.5bps, 4 150bps, 5 300bps, 6 600bps, 7 1200bps, 8 1800bps, 9 2400bps, 10 4800bps, 11 7200bps, 12 9600bps, 13 19200bps, 14 38400bps, 15 57600bps, 16 115200bps, 17 230400bps *mode* bit_cnt OR stop_bit OR parity. bit_cnt (bits 0-1) = 0x00 Data bit = 5; 0x01 Data bit = 6; 0x02 Data bit = 7; 0x03 Data bit = 8 stop_bit (bit 2) = 0x00 Stop bit = 1; 0x04 Stop bits = 1.5 or 2 parity (bits 3-5) = 0x00 No parity; 0x08 Odd parity; 0x18 Even parity; 0x28 Mark parity; 0x38 Space parity *flow* Flow control 0 None. 1 RTS/CTS. 2 XON/XOFF. 3 DTR/DSR.
Description	Set serial port parameter.
Return Value	0 O.K. -1 Error port number. -2 Error argument.

sysc_SetToDefault	Set to default value.
C Format	
Syntax	**#include <sdkconf.h>**
	void sysc_SetToDefault (void);
Arguments	N/A
Description	Restore all configurations back to factory default value.
Return Value	N/A

Flash ROM Access Library Reference

flash_erase	Erase flash-ROM.
C Format	
Syntax	**#include <sdkflash.h>**
	int flash_erase (void)
Arguments	N/A
Description	Erase flash-ROM.
Return Value	0 O.K.
	-1 Fail.

flash_length	Get current data length at the flash-ROM.
C Format	
Syntax	**#include <sdkflash.h>**
	long flash_length (void)
Arguments	N/A
Description	Get current data length at the flash-ROM
Return Value	>=0 Current data length at the flash-ROM. 0 after calling **flash_erase()**. Max. length is 163840. (160 KB).

flash_read	Read data from the flash-ROM.
C Format	
Syntax	**#include <sdkflash.h>**
	long flash_read (long *offset*, char **buffer*, long *size*);
Arguments	*offset* Indicates the point at which the function starts reading the buffer, measured in bytes.
	buffer Read buffer pointer.
	size Buffer size.
Description	Read data from the flash-ROM.
Return Value	>=0 Read data size.
	-1 Read failed.

flash_write	Write data to the flash-ROM.
C Format	
Syntax	**#include <sdkflash.h>**
	long flash_write (char ****buffer*, long *size*);**
Arguments	*buffer* Write data buffer pointer.
	size Write data size, from 1 to 163840.
Description	Write data to the flash-ROM.
Return Value	>0 Write length.
	-1 Write failed.
	-2 Need to erase the flash-ROM first.

sys_FlashErase	Erase flash ROM.
C Format	
Syntax	**#include <sdkflash.h>**
	int sys_FlashErase (int *band*);
Arguments	*band* Memory band, from 0 to 4
Description	Erase flash rom.
Return Value	-1 Fail.
	-2 Error argumnet.
	0 O.K.

sys_FlashLength	Get current data length at the flush-ROM.
C Format	
Syntax	**#include <sdkflash.h>**
	long sys_FlashLength (int *band*);
Arguments	*band* Memory bank, from 0 to 4.
Description	Get current data length at the flush-ROM.
Return Value	>=0 Current data length at the flush-ROM. 0 after call **sys_FlashErase()**. Maximum length is 32768. (32K bytes).
	-2 Error argument.

sys_FlashRead	Read data to flush-ROM.
C Format	
Syntax	**#include <sdkflash.h>**
	long sys_FlashRead (int *bank*, long *offset*, char * *buffer*, long *size*);
Arguments	*bank* Memory bank, from 0 to 4.
	offset Start read offset from bank begin.
	buffer Read buffer pointer.
	size Write data size, from 1 to 32768.
Description	Read data to flush-ROM.
Return Value	>=0 Size of data read..
	-1 Read fail.
	-2 Error argumnet.

sys_FlashWrite	Write data to flush-ROM.
C Format	
Syntax	**#include <sdkflash.h>**
	long sys_FlashWrite (int *bank*, char **buffer*, long *size*);
Arguments	*bank* Memory bank, from 0 to 4.
	buffer Write data buffer pointer.
	size Write data size, from 1 to 32768.
Description	Write data to flush-ROM.
Return Value	>=0 Length of data written.
	-1 Write fail.
	-2 Error argumnet.
	-3 Need erase flash-ROM first.

Debug Library Reference

dbg_put_block	Print out a block of data for debugging.
C Format	
Syntax	**#include <sdkdbg.h>**
	int dbg_put_block (char **buf*, int *len*);
Arguments	*buf* The print out debugging data buffer pointer.
	len Length of the debugging data buffer.
Description	Print out a block of data for debugging.
Return Value	This function returns the length of messages printed out.

dbg_printf	Print formatted output to the debug output stream.
C Format	
Syntax	**#include <sdkdbg.h>**
	int dbg_printf (char **format* [, *argument*]...);
Arguments	*format* Format control.
	argument Optional arguments.
Description	Print formatted output to the debug output stream.
Return Value	Each of these functions returns the number of characters printed, or a negative value if an error occurs.

dbg_put_ch	Print out a character for debugging.
C Format	
Syntax	**#include <sdkdbg.h>**
	int dbg_put_ch (char *ch*);
Arguments	*ch* The character value that will be printed out.
Description	Print out a character for debugging.
Return Value	This function returns the length of the printed out messages.

dbg_put_doubleword	Print out a 4-byte unsigned long value for debugging.
C Format	
Syntax	**#include <sdkdbg.h>**
	int dbg_put_doubleword (unsigned long *value*);
Arguments	*value* The printed out unsigned long value.
Description	Print out a 4-byte unsigned long value for debugging.
Return Value	This function returns the length of print out messages

dbg_put_doubleword_hex	Print out a 4-byte unsigned long value with HEX format for debugging.
C Format	
Syntax	**#include <sdkdbg.h>**
	int dbg_put_doubleword_hex (unsigned long *value*);
Arguments	*value* The printed out unsigned long value.
Description	Print out a 4-byte unsigned long value with HEX format for debugging.
Return Value	This function returns the length of print out messages.

dbg_put_IP	Print out an IP address in the a.b.c.d format for debugging.
C Format	
Syntax	**#include <sdkdbg.h>**
	int dbg_put_IP (unsigned long *ipaddr*);
Arguments	*ipaddr* The printed out Internet host's IP address.
Description	Print out an IP address in the a.b.c.d format for debugging.
Return Value	This function returns the length of print out messages.

dbg_put_string	Print out a string for debugging.
C Format	
Syntax	**#include <sdkdbg.h>**
	int dbg_put_string (char **buf*);
Arguments	*buf* The printed out debugging data buffer's pointer.
Description	Print out a string for debugging.
Return Value	This function returns the length of print out messages.

dbg_put_word	Print out a 2-byte unsigned integer value for debugging.
C Format	
Syntax	**#include <sdkdbg.h>**
	int dbg_put_word (unsigned short *value*);
Arguments	*value* The printed out unsigned short value.
Description	Print out a 2-byte unsigned integer value for debugging.
Return Value	This function returns the length of print out messages.

dbg_put_word_hex	Print out a 2-byte unsigned integer value with HEX format for debugging.
C Format	
Syntax	**#include <sdkdbg.h>**
	int dbg_put_word_hex (unsigned short *value*);
Arguments	*value* The print out unsigned short value.
Description	Print out a 2-byte unsigned integer value with HEX format for debugging.
Return Value	This function returns the length of print out messages.

Thread Control Library Reference

sys_ThreadClose	Close a thread.
C Format	
Syntax	**#include <sdktask.h>**
	int sys_ThreadClose (unsigned long *threadID*);
Arguments	*threadID* Thread ID returned from **sys_ThreadCreate()**.
Description	Close a thread.
Return Value	0 O.K.
	-1 Error *threadID*.

sys_ThreadCreate	Create a thread.
C Format	
Syntax	**#include <sdktask.h>**
	int sys_ThreadCreate (void (ap*) (unsigned long *arg*), char **stack_ptr*, long *stack_size*, unsigned long *parameter*, int *createFlags*, unsigned long **threadIDp*);**
Arguments	*ap* This thread entry pointer.
	stack_ptr This thread stack pointer.
	stack_size This thread stack size.
	parameter The argument passing to **ap()**.
	createFlags Currently, this is no used, must be 0.
	threadIDp Pointer to a buffer to return the new thread ID.
Description	Create a thread.
Return Value	0 O.K.
	-1 No resource.
	-2 Error argument.

sys_ThreadResume	Resume a thread.
C Format	
Syntax	**#include <sdktask.h>**
	int sys_ThreadResume (unsigned long *threadID*);
Arguments	*threadID* Thread ID returned from **sys_ThreadCreate()**.
Description	Resume a thread.
Return Value	0 O.K.
	-1 Error *threadID*.

sys_ThreadState	Get a thread state.
C Format	
Syntax	**#include <sdktask.h>**
	int sys_ThreadState (unsigned long *threadID*);
Arguments	*threadID* Thread ID returned from **sys_ThreadCreate()**.
Description	Get a thread state.
Return Value	1 Suspend
	0 Running
	-1 Error *threadID*.

sys_ThreadSuspend	Suspend a thread.
C Format	
Syntax	**#include <sdktask.h>**
	int sys_ThreadSuspend (unsigned long *threadID*);
Arguments	*threadID* Thread ID returned from **sys_ThreadCreate()**.
Description	Suspend a thread.
Return Value	0 O.K.
	-1 Error *threadID*.

Time Server Library Reference

sys_GetLocalTime	Get local time.
C Format	
Syntax	**#include <sdksys.h>**
	int sys_GetLocalTime (struct tm_local **tm*);
Arguments	*tm* Local time information.
Description	Get local time.
Return Value	0 OK.
	-1 Fail.

sys_SetLocalTime	Set local time.
C Format	
Syntax	**#include <sdksys.h>**
	int sys_SetLocalTime (struct tm_local **tm*);
Arguments	*tm* Time information that you want to set.
Description	Set local time.
Return Value	0 OK.
	-1 Fail.

sysc_getTimeServer	Return current time server used to syncronize the system time.
C Format	
Syntax	**#include <sdkconf.h>**
	int sysc_getTimeServer (char *_buffer_, int _bufsize_);
Arguments	*buffer* Pointer to the buffer to retrieve the time server address.
	bufsize The size of *buffer*, in bytes.
Description	Return current time server used to syncronize the system time.
Return Value	>= 0 Number of bytes placed in *buffer*.
	-1 Fail.

sysc_getTimeZone	Get the time offset used by time syncronization.
C Format	
Syntax	**#include <sdkconf.h>**
	long sysc_getTimeZone (void);
Arguments	N/A
Description	Retrieve the time offset from local time zone to UTC. The offset is used while time syncronization. The returned value is the time offset to UTC, in seconds. e.q. a GMT +8:00 time zone has 28800 seconds offset.
Return Value	>= 0 Time offset from local time to UTC.

sysc_getTZoneIndex	Get the time zone of local system.
C Format	
Syntax	**#include <sdkconf.h>**
	int sysc_getTZoneIndex(void);
Arguments	N/A
Description	Retrieve the time zone of local system. All time zones are listed later in this chapter.
Return Value	>= 0 Time zone index number.
	-1 Fail.

sysc_setTimeServer	Set time server that is used to syncronize the system time.
C Format	
Syntax	**#include <sdkconf.h>**
	int sysc_setTimeServer (char *_buffer_, int _bufsize_);
Arguments	*buffer* The new time server address.
	bufsize The length of server address in *buffer*, in bytes.
Description	NPort can syncronize its system time with a remote NTP server. To enable this function, the NTP server address must be specified.
Return Value	>= 0 Number of bytes copied from *buffer* to system configuration.
	-1 Fail.

sysc_setTimeZone	Set the time offset used by time syncronization.
C Format	
Syntax	**#include <sdkconf.h>** **void sysc_setTimeZone (long *tz*);**
Arguments	*tz* The time offset from local time zone to UTC.
Description	Set the time offset from local time zone to UTC. The offset will be used while time syncronization. The offset is mesured in seconds. e.q. set this value to 28800 for a GMT +8:00 time zone.
Return Value	N/A

sysc_setTZoneIndex	Set the time zone of local system.
C Format	
Syntax	**#include <sdkconf.h>** **void sysc_setTZoneIndex (int *index*);**
Arguments	*index* Time zone index number.
Description	Set the time zone of local system. All time zones are listed later in this chapter. Note that time zone indicated by *index* is used for human readabilty and takes no effect when syncronizing the local time with the time server. You should call **sysc_setTimeZone()** to set the actual time offset.then.
Return Value	N/A

Time Zone Offsets Index

The hour offsets for different time zones are listed below. You will need this information when setting the time zone in 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

Index	Offset	Status	Region
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
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

Index	Offset	Status	Region
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.

Memory Management Library Reference

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

sys_free	Deallocates or frees a memory block.
C Format	
Syntax	**#include <sdksys.h>**
	void sys_free (void **ptr*);
Arguments	*ptr* Pointer to a memory returned by **sys_malloc()** or **sys_calloc()**.
Description	The **sys_free()** function causes the allocated memory referenced by *ptr* to be made available for future allocations.
Return Value	N/A

sys_getFreeMemSize	Get available memory status used by **sys_malloc()** and **sys_calloc()**.
C Format	
Syntax	**#include <sdksys.h>** **int sys_getFreeMemSize (long **total_size*, long **max_block_size*);**
Arguments	*total_size* Pointer to a buffer to retrieve total size of free memory, in bytes. *max_block_size* Pointer to a buffer to retrieve the maximum size of free memory blocks.
Description	When user program requests to allocate memory by **sys_malloc()** or so on, system will return a block of continuous free memory. After times of allocation and deallocation, the entire memory space may contain several small blocks of free memory with different size. When total free memory may still large enough, **sys_malloc()** may fail if there is no continuous memory block large enough to keep the allocated size. After **sys_getFreeMemorySize()** is called, the total size of free memory is put in the buffer pointed by *total_size*, and the maximum size of free memory blocks is put in the buffer pointed by *max_block_size*. This function is useful for debugging. You can use it to determine if memory leak occurs.
Return Value	0 OK.

sys_malloc	Allocates memory blocks.
C Format	
Syntax	**#include <sdksys.h>** **void *sys_malloc (unsigned *size*);**
Arguments	*size* Number of elements to be allocated.
Description	The **sys_malloc()** function allocates *size* bytes of memory. The allocated space is suitably aligned (after possible pointer coercion) for storage of any type of object. Note that **sys_malloc()** does NOT normally initialize the returned memory to zero bytes.
Return Value	The **sys_malloc()** function returns a pointer to the allocated memory if successful; otherwise a NULL pointer is returned.

sys_realloc	Reallocate memory blocks.
C Format	
Syntax	**#include <sdksys.h>** **void *sys_realloc (void **ptr*, unsigned *newsize*);**
Arguments	*ptr* Pointer to previously allocated memory block. *newsize* New size in bytes.
Description	The **sys_realloc()** function changes the size of the previously allocated memory referenced by *ptr* to *newsize* bytes. The contents of the memory are unchanged up to the lesser of the new and old sizes. If the new size is larger, the value of the newly allocated portion of the memory is undefined. If the requested memory cannot be allocated, NULL is returned and the memory referenced by *ptr* is valid and unchanged.
Return Value	The **sys_realloc()** function returns a pointer, possibly identical to *ptr*, to the allocated memory if successful; otherwise a NULL pointer is returned. The **sys_realloc()** function always leaves the original buffer intact when an error occurs.

4

External Function Calls for SDK

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

Function Name	Description
strcat()	Append a string.
strchr()	Find a character in a string.
strcmp()	Compare strings.
strcpy()	Copy a string.
strlwr()	Convert a string to lowercase.
strupr()	Convert a string to uppercase.
strlen()	Get the length of a string.
atoi()	Convert strings to integer.
atol()	Convert strings to long.
itoa()	Convert an integer to a string.
ltoa()	Convert a long integer to a string.

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

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

NOTE *Keep in mind that you must use the complete path to link to the cl.lib library file.*

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

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

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