Windows Standard

Serial Communications

Reference Library

(WSC_REF)

Version 4.4

January 19, 2009

This software is provided as-is.
There are no warranties, expressed or implied.

Copyright (C) 1996 - 2009
All rights reserved

MarshallSoft Computing, Inc.
Post Office Box 4543
Huntsville AL 35815 USA

Voice : 1.256.881.4630
Email : info@marshallsoft.com
Web : www.marshallsoft.com

MARSHALLSOFT is a registered trademark of MarshallSoft Computing.

TABLE OF CONTENTS

1 Introduction

The **Windows Standard Serial Communications Library** (**WSC)** is a serial communication component DLL library that provides full control over a serial port. WSC uses the standard Windows API (Application Programmer's Interface) to communicate with any device connected to a serial port.

A simple interface allows accessing data from a serial port using RS232 or multi-drop RS422 / RS485 serial ports. **Windows Standard Serial Communications Library** (**WSC)** also supports virtual ports such as those created by Bluetooth and USB/serial converters.

The WSC Reference Manual (WSC_REF) applies to the **Windows Standard Serial Communications Library** (WSC) for all supported languages. It contains details on each individual WSC function.

1.1 General Remarks

All functions return an integer code. Negative values are always errors. See "WSC Error Codes" in Section 5.1. Non-negative (>=0) return codes are never errors.

Each function argument is marked as:

- `(I) : 2-byte integer (Win16), 4-byte integer (Win32).`
- `(S) : 2-byte short integer (Win16 and Win32).`
- `(L) : 4-byte integer (Win16 and Win32).`
- `(P) : 4-byte pointer (Win16 and Win32).`

Refer to the declaration files (see Section 1.3 below) for the exact syntax of each WSC function. Also note that the example programs, found in the /APPS directory, show exactly how WSC functions are called.

The latest version of our serial comm software and complete technical documentation can be found online at **`http://www.marshallsoft.com/serial-communication-library.htm`**

1.2 Documentation Set

The complete set of documentation consists of four manuals in two formats. This is the third manual (WSC_REF) in the set.

- WSC 4x Programmer's Manual (`WSC_4x.PDF and WSC_4x.HTM`)
- WSC User's Manual (`WSC_USR.PDF and WSC_USR.HTM`)
- WSC Reference Manual (`WSC_REF.PDF and WSC_REF.HTM`)
- SERIAL User's Manual (`SERIAL.PDF and SERIAL.HTM`)

Each manual comes in two formats:

- Adobe PDF (files ending in .PDF). The best format for printing manuals.
- Hyper Text (files ending in .HTM). Use any web browser to read.

The WSC_4x Programmer's Manual is the language dependent manual and provides information needed to compile your programs as well as the examples in the specified environment. The "x" in WSC_4x Programmer's Manual specifies the host language such as C for C/C++, VB for Visual Basic, etc.

The WSC User's Manual (WSC_USR) discusses language independent serial communications programming issues including modem control. It also contains purchase and license information. The WSC Reference Manual (WSC_REF) contains details on each individual WSC function.

The Serial Communications Manual (SERIAL) contains background information on serial port hardware.

1.3 Declaration Files

The exact syntax for calling WSC functions are specific to the host language (C/C++, Delphi, VB, etc.) and are defined for each language in the “WSC declaration files”. Each WSC product comes with the appropriate declaration file for the supported language. For example,

```
WSC4C   C/C++,NET,C#              WSC.H
WSC4VB  Visual Basic              WSC16.BAS and WSC32.BAS
        VB.NET                    WSC32.VB
        VBA (EXCEL,ACCESS,etc.)   WSC16.BAS and WSC32.BAS
WSC4PB  PowerBASIC                WSC32.PBI
WSC4D   Borland Delphi            WSC16.PAS and WSC32.PAS
WSC4CB  Fujitsu COBOL             WSC32.CBI
WSC4FP  Visual FoxPro             WSC32.FOX
WSC4DB  Visual dBase              WSC16.CC and WSC32.CC
WSC4XB  Xbase++                   WSC32.CH
```

1.4 Language Notes

All language versions of **Windows Standard Serial Communications Library** (**WSC)** include the example program WSCVER. Refer to this program and the declaration file as defined in Section 1.3 above to see how WSC functions are called.

The best way to see how a function is called is to find it used in one of the example programs. All WSC functions are used in one or more examples.

1.4.1 C/C++/C# (and .NET)

None.

1.4.2 Delphi

(1) Functions defined in the Delphi Unit WSCW.PAS begin with "f" rather than "Sio".
(2) Replace "=" with ":=" in the examples.

1.4.3 Visual Basic (and VB.NET)

None.

1.4.4 PowerBASIC

(1) Constants defined for PowerBASIC (WSC32.PBI) begin with the character '%' symbol.
(2) The WSC keycode is defined in KEYCODE.PBI.

1.4.5 Visual FoxPro

All strings passed to WSC functions must be prefixed with the '@' character.

1.4.6 Visual dBase

None.

1.4.7 Xbase++

(1) Functions defined for Xbase++ begin with 'X'.
(2) All strings passed to WSC functions must be prefixed with the '@' character.

2 WSC Functions

2.1 SioBaud :: Sets the baud rate.

SYNTAX

```
SioBaud(Port, Baud)

   Port : (I) -1 or port selected.
   Baud : (I) Baud code or actual baud rate.
```

REMARKS

The **SioBaud** function sets the baud rate without resetting the port. It is used to change the baud rate after calling **SioReset. SioBaud** may be called with either the actual baud rate value or one of the baud rate codes as follows:

[VALUE]	[RATE]	[NAME]
0	110	Baud110
1	300	Baud300
2	1200	Baud1200
3	2400	Baud2400
4	4800	Baud4800
5	9600	Baud9600
6	19200	Baud19200
7	38400	Baud38400
8	57600	Baud57600
9	115200	Baud115200

Note that the baud rate does NOT have to be one listed above. When **SioReset** is called, the baud rate is set to 19200 until changed by calling **SioBaud**. The 19200 default baud rate can be changed by calling **SioBaud** with Port set to -1 before calling **SioReset**. Subsequent calls to **SioReset** will then use the new default baud rate.

EXAMPLE

```
Code = SioBaud(COM1, 28800)
```

RETURNS

- `Return = WSC_IE_BADID` (No such port)
- `Return = WSC_IE_BAUDRATE` (Unsupported baud rate)

2.2 **SioBrkSig** :: **Asserts**, cancels, or detects BREAK signal.

SYNTAX

```
SioBrkSig(Port, Cmd)

  Port : (I) Port selected.
  Cmd  : (I) ASSERT, CANCEL, or DETECT.
```

REMARKS

The **SioBrkSig** function controls the BREAK bit in the line status register. The legal commands are:

```
[NAME]             :  [FUNCTION]
WSC_ASSERT_BREAK   :  to assert BREAK
WSC_CANCEL_BREAK   :  to cancel BREAK
WSC_DETECT_BREAK   :  to detect BREAK
```

WSC_ASSERT_BREAK, WSC_CANCEL_BREAK, and WSC_DETECT_BREAK are defined in the language declaration file (see Section 1.3).

RETURNS

- `Return = WSC_IE_NOPEN` (Port not opened. Call **SioReset** first)
- `Return = WSC_RANGE` (Illegal command. Expected 'A', 'C', or 'D')

EXAMPLE

```
Code = SioBrkSig(Port, WSC_ASSERT_BREAK)
```

2.3 **SioByteToShort** :: Converts 8-bit Character Buffer to 16-bit Unicode ASCII

SYNTAX

```
SioByteToShort(Buffer)

  Buffer : (P) character buffer
```

REMARKS

The **SioByteToShort** function converts the (null terminated) character buffer 'Buffer' from 8-bit ASCII characters to 16-bit Unicode ASCII characters.

The buffer must be null terminated (last character is a hex 00) and the buffer must be at least twice the size (in bytes) of the character string (since 16-bit characters require twice the space as 8-bit characters).

This function is only necessary when working with 16-bit Unicode ASCII characters.

RETURNS

None.

EXAMPLE (C#)

```
char[] UnsafeBuffer = new char[128];
// get the registration string
fixed (char* pBuffer = UnsafeBuffer)
Code = SioGetReg(pBuffer, 50);
if(Code>0)
  {// convert (null terminated) UnsafeBuffer[] to 16-bit chars (unicode)
   fixed (char* pBuffer = UnsafeBuffer)
   SioByteToShort(pBuffer);
  }
```

ALSO SEE

SioShortToByte

2.4 **SioCTS** :: Reads the Clear to Send (CTS) modem status bit.

SYNTAX

```
SioCTS(Port)

   Port : (I) Port selected.
```

REMARKS

The **SioCTS** function is used to detect if CTS (Clear To Send) is set (1) or clear (0). Some Windows Win16 COMM drivers cannot read the CTS bit correctly!

The CTS line is used by some error correcting modems to implement hardware flow control. CTS is dropped by the modem to signal the computer not to send data and is raised to signal the computer to continue.

Refer to the SERIAL User's Manual (SERIAL.HTM) for a discussion about flow control.

RETURNS

- `Return = WSC_IE_NOPEN` (Port not opened. Call **SioReset** first)
- `Return = WSC_IE_BADID` (No such port)
- `Return = 0` (CTS is clear)
- `Return > 0` (CTS is set)

EXAMPLE

```
Code = SioCTS(Port)
```

ALSO SEE

See SioFlow and SioRead.

2.5 **SioDCD** :: Reads the Data Carrier Detect (DCD) modem status bit

SYNTAX

```
SioDCD(Port)

   Port : (I) Port selected.
```

REMARKS

The **SioDCD** function is used to read the Data Carrier Detect (DCD) modem status bit. Also see **SioStatus**.

SioDCD is normally used after connecting to check that the carrier has not been dropped.

RETURNS

- `Return = WSC_IE_NOPEN` (Port not opened. Call **SioReset** first)
- `Return = WSC_IE_BADID` (No such port)
- `Return = 0` (DCD is clear)
- `Return > 0` (DCD is set)

EXAMPLE

```
Code = SioDCD(Port)
```

ALSO SEE

See SioRead.

2.6 **SioDebug** :: Sets and/or reads debug data.

SYNTAX

```
SioDebug(Parm)

   Parm : (I) Parameter.
```

REMARKS

Passing the character 'R' will result in the serial port driver RESETDEV ("reset device") command being called when **SioReset** is called. The RESETDEV command is not required for the operation of the UART and is not always implemented by some serial devices such as USB-Serial adapters.

Passing the character 'W' will toggle the operation of **SioPuts** between (1) "wait for completion" [default] and (2) "immediate return" modes, as described in Section-2.9 of the WSC User's Manual (WSC_USR.PDF) or http://www.marshallsoft.com/wsc_usr.htm#Section_2.9

RETURNS

See remarks above.

EXAMPLE

C++ Example

```
Code = SioDebug('W');
```

BASIC Example

```
Code = SioDebug(ASC("W"))
```

ALSO SEE

None.

2.7 **SioDone** :: Terminates further serial processing.

SYNTAX

```
SioDone(Port)

   Port : (I) Port selected.
```

REMARKS

The **SioDone** function terminates further serial port processing, allowing other applications to use the port. **SioDone** should always be the last function called before exiting an application.

If an application is running from within an integrated development environment (IDE) and the application terminates without **SioDone** being called first, the IDE itself will prevent the port from being re-opened. Terminating the IDE will free the port.

RETURNS

- `Return = WSC_IE_NOPEN` (Port not opened. Call **SioReset** first)
- `Return = WSC_IE_BADID` (No such port)

EXAMPLE

```
Code = SioDone(Port)
```

ALSO SEE

See SioReset.

2.8 **SioDSR** :: Reads the Data Set Ready (DSR) modem status bit.

SYNTAX

```
SioDSR(Port)

   Port : (I) Port selected.
```

REMARKS

The **SioDSR** function is used to detect if DSR (Data Set Ready) is set (1) or clear (0). Some Windows Win16 COMM drivers cannot read the DSR bit correctly!

Modems normally set DSR as soon as they are powered up.

RETURNS

- `Return = WSC_IE_NOPEN` (Port not opened. Call **SioReset** first)
- `Return = WSC_IE_BADID` (No such port)
- `Return = 0` (DSR is clear)
- `Return > 0` (DSR is set)

EXAMPLE

```
     Code = SioDSR(Port)
```

ALSO SEE

See SioRead.

2.9 **SioDTR** :: Set, clear, or read Data Terminal Ready (DTR).

SYNTAX

```
SioDTR(Port, Cmd)

   Port : (I) Port selected.
   Cmd  : (I) DTR command (see below).
```

REMARKS

The **SioDTR** function controls the Data Terminal Ready (DTR) bit in the modem control register. DTR should always be set when communicating with a modem.

```
[NAME]          :  [FUNCTION]
WSC_SET_LINE    :  to set DTR (ON)
WSC_CLEAR_LINE  :  to clear DTR (OFF)
WSC_READ_LINE   :  to read DTR
```

RETURNS

- `Return = WSC_IE_NOPEN` (Port not opened. Call **SioReset** first)
- `Return = WSC_IE_BADID` (No such port)
- `Return = WSC_RANGE` (Not one of 'S', 'C', or 'R')
- `Return = 0` (DTR is clear [READ_LINE Command])
- `Return >0` (DTR is set [READ_LINE Command])

EXAMPLE

```
Code = SioDTR(Port, WSC_SET_LINE)
```

ALSO SEE

SioRead.

2.10 **SioEvent** :: Efficiently waits for serial event.

SYNTAX

```
SioEvent(Port, Mask)

   Port : (I) Port selected.
   Mask : (I) Event Mask (see below).
```

REMARKS

The **SioEvent** function (WIN32 only) waits (blocks) until the condition specified in 'Mask' is satisfied. **SioEvent** returns (unblocks) only for events that occur after it is called. Multiple conditions can be OR'ed together. See example below. The event masks are:

```
[NAME]      :  [FUNCTION]
EV_RXCHAR   :  A character was received.
EV_BREAK    :  A break signal was received.
EV_CTS      :  The CTS line changed states.
EV_DSR      :  The DSR line changed states.
EV_ERR      :  An error was detected.
EV_RLSD     :  The DCD line has changed states.
EV_RING     :  The RI line has been set.
EV_TXEMPTY  :  The TX queue has become empty.
```

Call the SioEventWait function.if an event timeout is desired.

RETURNS

SioEvent does not return until the specified event occurs. For this reason, it is best used inside of a thread.

- `Return = WSC_IE_NOPEN` (Port not opened. Call **SioReset** first)
- `Return = WSC_IE_BADID` (No such port)
- `Return = WSC_IO_ERROR` (An event error has occurred)
- `Return = WSC_IO_COMPLETE` (success - event has occurred)
- `Return = WSC_IO_PENDING` (fails - event has not occurred)

WSC_IO_PENDING will be returned by **SioEvent** if timeout has occurred.

EXAMPLE

C/C++ Example

```
// wait until CTS or DSR changes states.
Code = SioEvent(Port, EV_CTS|EV_DSR)
```

BASIC Example

```
//   ' wait until CTS or DSR changes states.
     Code = SioEvent(Port, EV_CTS OR EV_DSR)
```

ALSO SEE

SioEventChar, SioEventWait, and SioMessage.

2.11 **SioEventChar** :: Efficiently waits for a serial character.

SYNTAX

```
SioEventChar(Port, EvtChar, Timeout)

    Port : (I) Port selected.
 EvtChar : (I) Event character.
 Timeout : (I) Timeout (milliseconds).
```

REMARKS

The **SioEventChar** function (WIN32 only) waits (blocks) until the specified character is seen in the serial input stream or timeout occurs. **SioEventChar** returns (unblocks) only when the specified character is received after it is called.

RETURNS

- `Return = WSC_IE_NOPEN` (Port not opened. Call **SioReset** first)
- `Return = WSC_IE_BADID` (No such port)
- `Return = WSC_IO_ERROR` (An event error has occurred)
- `Return = WSC_IO_COMPLETE` (success - event has occurred)
- `Return = WSC_IO_PENDING` (fails - event has not occurred)

WSC_IO_PENDING will be returned by **SioEventChar** if timeout has occurred.

EXAMPLE

C/C++ Example

```
// wait (up to 1 second) until a carriage return is seen.
Code = SioEventChar(Port, '\r', 1000)
```

BASIC Example

```
' wait (up to 1 second) until a carriage return [ Chr(13) ] is seen.
Code = SioEventChar(Port, 13, 1000)
```

ALSO SEE

SioEvent and SioEventWait.

2.12 **SioEventWait** :: Efficiently waits for a serial event.

SYNTAX

```
SioEventWait(Port, Mask, Timeout)

    Port : (I) Port selected.
    Mask : (I) Event Mask (see below).
 Timeout : (I) Timeout (milliseconds).
```

REMARKS

The **SioEventWait** function (WIN32 only) waits (blocks) until the condition specified in 'Mask' is satisfied or the timeout is reached. **SioEventWait** returns (unblocks) only for events that occur after it is called unless the specified timeout period is reached. Multiple conditions can be OR'ed together. See the example below. The event masks can be found in the description of **SioEvent** entry above.

RETURNS

- `Return = WSC_IE_NOPEN` (Port not opened. Call **SioReset** first)
- `Return = WSC_IE_BADID` (No such port)
- `Return = WSC_IO_ERROR` (An event error has occurred)
- `Return = WSC_IO_COMPLETE` (success - event has occurred)
- `Return = WSC_IO_PENDING` (fails - event has not occurred)

WSC_IO_PENDING will be returned by **SioEventWait** if timeout has occurred.

EXAMPLE

C/C++ Example

```
// Wait (up to 1.5 seconds) for incoming serial data.
Code = SioEventWait(Port, EV_RXCHAR, 1500)
```

BASIC Example

```
' Wait (up to 1.5 seconds) for incoming serial data.
Code = SioEventWait(Port, EV_RXCHAR, 1500)
```

ALSO SEE

SioEvent and SioEventChar.

2.13 **SioFlow** :: Sets flow control protocol.

SYNTAX

```
SioFlow(Port, Cmd)

   Port : (I) Port selected.
   Cmd  : (I) Class of flow control (see below).
```

REMARKS

The **SioFlow** function is used to enable or disable hardware flow control. Hardware flow control uses RTS and CTS to control data flow between the modem and the computer. To enable flow control, call **SioFlow** with 'Cmd' set to:

```
[NAME]                      :  [FUNCTION]
WSC_HARDWARE_FLOW_CONTROL   :  Hardware (RTS/CTS) flow control.
WSC_SOFTWARE_FLOW_CONTROL   :  Software (XON/XOFF) flow control.
WSC_NO_FLOW_CONTROL         :  No flow control [default].
```

In order for flow control to work correctly, your serial device must also be configured to work with the same class of flow control (hardware or software). If using hardware flow control, the computer to serial device cable must have RTS and CTS wired straight through. If hardware flow control is enabled, the RTS line should not be modified by calling **SioRTS**.

RETURNS

- `Return = WSC_RANGE` (Cannot recognize command)
- `Return > 0` (Flow control enabled)
- `Return = 0` (Flow control disabled)

EXAMPLE

```
Code = SioFlow(Port, WSC_HARDWARE_FLOW_CONTROL)
```

ALSO SEE

SioPutc and SioSetTimeouts.

2.14 **SioGetc** :: Reads the next character from the serial line.

SYNTAX

```
SioGetc(Port)

   Port : (I) Port selected.
```

REMARKS

The **SioGetc** function reads the next byte from the receive queue of the selected serial port. WSC_NO_DATA (-100) is returned if no byte is available.

RETURNS

- `Return = WSC_IE_NOPEN` (Port not opened. Call **SioReset** first)
- `Return = WSC_IE_BADID` (No such port)
- `Return = WSC_NO_DATA` (no data available)
- `Return >= 0` (character read)

EXAMPLE

```
     Code = SioGetc(Port)
```

ALSO SEE

SioUnGetc and SioGets.

2.15 **SioGetReg** :: Returns the license registration string.

SYNTAX

```
SioGetReg(Buffer, BufLen)

   Buffer : (P) Buffer for registration string.
   BufLen : (I) Length of above buffer.
```

REMARKS

The **SioGetReg** function copies the license registration string (a maximum of 50 bytes) to ‘Buffer’.

The registration string identified the purchaser and is embedded in each registered DLL.

RETURNS

```
Number of bytes copied.
```

EXAMPLE

```
     Length = SioGetReg (Buffer, 50)
```

2.16 **SioGets** :: Reads the next byte buffer from the serial line.

SYNTAX

```
SioGets(Port, String, Cnt)

   Port   : (I) Port selected.
   String : (P) Pointer to receive buffer.
   Cnt    : (I) Number of bytes to read.
```

REMARKS

The **SioGets** function reads the smaller of the number of bytes wanted (Cnt) and the number of bytes in the receive buffer. A zero is returned if no bytes are read.

Note that even if the data is being sent in one operation by the other side, it may not necessarily arrive all at in a single packet.

RETURNS

- `Return = WSC_IE_NOPEN` (Port not opened. Call **SioReset** first)
- `Return = WSC_IE_BADID` (No such port)
- `Return >= 0` (Number of characters actually read)

EXAMPLE

`C/C++ Example`

```
char Buffer[128];
Code = SioGets(Port, (LPSTR)Buffer, 128)
```

`BASIC Example`

```
Dim Buffer As String * 128
Code = SioGets(Port, Buffer, 128)
```

ALSO SEE

SioUnGetc and SioPutc.

2.17 **SioInfo** :: Returns WSC library version information.

SYNTAX

```
SioInfo(Cmd)

   Cmd : (I) Command (See below)
```

REMARKS

The **SioInfo** function returns an integer code corresponding to the Cmd as follows.

```
[NAME]            :  [FUNCTION]
WSC_GET_VERSION   :  Library version number [3 hex digits].
WSC_GET_BUILD     :  Library build number.
```

`SioInfo``(WSC_GET_VERSION)` will return the 3 digit version number embedded in WSC16.DLL and in WSC32.DLL. The 3 digit version number is formatted as the rightmost 3 nibbles (4 bits per nibble) of the return value. **`SioInfo`**`(WSC_GET_BUILD)` will return the version build number.

Refer to the WSCVER program for an example.

RETURNS

See remarks above.

`Return = -1` (Cannot recognize command)

EXAMPLE

```
Code = SioInfo(WSC_GET_VERSION)
```

2.18 **SioKeyCode** :: Pass keycode to WSC32.DLL

SYNTAX

```
SioKeyCode(KeyCode)

  KeyCode : (L) Keycode value (0 or 8 to 10 digit number)
```

REMARKS

The **SioKeyCode** function must be the first **WSC** call made.

When WSC is registered, you will receive a 'keycode' (an 8 to 10 digit number) that matches the 'keycode' within the registered version of the DLL. For the evaluation (shareware) version, the keycode is 0. See file KEYCODE.

EXAMPLE

All example programs call **SioKeyCode**

```
    Code = SioKeyCode(WSC_KEY_CODE)
```

RETURNS

```
Return >= 0 No error.
Return = WSC_KEYCODE (wrong keycode)
```

2.19 **SioMessage** :: Send windows message when event occurs.

SYNTAX

```
SioMessage(Port, Handle, Message, Mask)

   Port   : (I) Port selected.
   Handle : (S) Window handle (HWND).
   Message: (I) Message (Usually WM_USER).
   Mask   : (L) Event mask (see SioEvent).
```

REMARKS

The **SioMessage** function will post the message 'Message' to the window handle 'Handle' when event 'Mask' occurs. **SioMessage** does not block.

Call **SioMessage**(Port, 0, 0, 0) in order to cancel a previous event.

Refer to **SioEvent** for a list of mask values.

RETURNS

See remarks above.

EXAMPLE

```
     Code = SioMessage(Port, hWnd, WM_USER, EV_RXCHAR)
```

ALSO SEE

`SioEvent, SioEventChar, and SioEventWait`

2.20 **SioParms** :: Sets parity, stop bits, and word length.

SYNTAX

```
SioParms(Port, Parity, StopBits, DataBits)

   Port     : (I) -1 or port selected.
   Parity   : (I) Parity code.
   StopBits : (I) Stop bits code.
   DataBits : (I) Word length code.
```

REMARKS

The **SioParms** function sets the parity, stop bits, and word length values.

SioParms can be called either before or after calling **SioReset**. Call **SioParms** with Port set to -1 before calling **SioReset** to make the passed parameters the default. Use the constant values defined in the WSC declaration file (see Section 1.3) to minimize the chance of passing an incorrect parameter value.

[PARITY]	[STOPBITS]	[DATABITS]
NoParity	OneStopBit	WordLength7
OddParity	One5StopBits	WordLength8
EvenParity	TwoStopBits	--
SpaceParity	--	--
MarkParity	--	--

RETURNS

- `Return = WSC_IE_BADID` (No such port)
- `Return = WSC_IE_BYTESIZE` (Word length not supported)
- `Return = WSC_RANGE` (Parameter out of range)

EXAMPLE

```
Code = SioParms(Port, WSC_NoParity, WSC_OneStopBit, WSC_WordLength8)
```

ALSO SEE

SioReset.

2.21 **SioPutc** :: Transmit a character over a serial line.

SYNTAX

```
SioPutc(Port, Ch)

   Port : (I) Port selected.
   Ch   : (I) Character to send.
```

REMARKS

The **SioPutc** function copies the character to the transmit queue for subsequent transmission by the UART.

RETURNS

- `Return = WSC_IE_NOPEN` (Port not opened. Call **SioReset** first)
- `Return = WSC_IE_BADID` (No such port)
- `Return = 1` (No error)

EXAMPLE

`C/C++ Example`

```
     Code = SioPutc(Port, 'A')
```

`BASIC Example`

```
     Code = SioPutc(Port, ASC("A"))
```

ALSO SEE

SioGetc, SioFlow, and SioSetTimeouts.

2.22 **SioPuts** :: Transmits a byte buffer over a serial line.

SYNTAX

```
SioPuts(Port, String, Count)

   Port   : (I) Port selected.
   String : (P) Pointer to string of bytes to transmit.
   Count  : (I) Number of bytes to transmit.
```

REMARKS

The **SioPuts** function copies 'Count' bytes from 'String' to the transmit queue for transmission. The 'String' can contain any ASCII or binary values.

The **SioPuts** function can operate in two ways: "wait for completion" and "immediate return", as described in Section-2.9 of the WSC User's Manual (WSC_USR.PDF) or
http://www.marshallsoft.com/wsc_usr.htm#Section_2.9

RETURNS

- `Return = WSC_IE_NOPEN` (Port not opened. Call **SioReset** first)
- `Return = WSC_IE_BADID` (No such port)
- `Return >= 0` (The number of bytes accepted for transmission)

EXAMPLE

C/C++ Example

```
char Buffer[128];
Code = SioPuts(Port, (LPSTR)Buffer, 128)
```

BASIC Example

```
Dim Buffer As String * 128
Code = SioPuts(Port, Buffer, 128)
```

ALSO SEE

SioGetc, SioFlow, and SioSetTimeouts.

2.23 **SioRead** :: Reads any UART register.

SYNTAX

```
SioRead(Port, Reg)

   Port : (I) Port selected.
   Reg  : (I) UART register (0 to 7).
```

REMARKS

SioRead is **ONLY** for Win16 applications running under Windows 95 and 98. The **SioRead** function reads any of the 7 I/O ports directly, by-passing the Windows API. The line status and the modem status registers can be read by (in C/C++):

```
#define SioLine(Port) SioRead(Port,5)
#define SioModem(Port) SioRead(Port,6)
```

Note that all modem and/or line status bits can also be read using **SioDTR**, **SioRTS**, **SioDCD**, **SioRI**, **SioDSR**, and **SioCTS**. Refer to the SERIAL User's Manual (SERIAL.PDF or http://www.marshallsoft.com/serial.htm) for a discussion of the UART registers.

RETURNS

- `Return = WSC_IE_NOPEN` (Port not opened. Call **SioReset** first)
- `Return = WSC_IE_BADID` (No such port)
- `Return = >0` (Contents of selected UART register)

EXAMPLE

```
Code = SioRead(Port, 5)
```

ALSO SEE

SioStatus.

2.24 **SioReset** :: Initialize a serial port for processing.

SYNTAX

```
SioReset(Port, RxQueSize, TxQueSize)

   Port      : (I) Port selected (or -1: see below).
   RxQueSize : (I) Receive queue size.
   TxQueSize : (I) Transmit queue size.
```

REMARKS

The **SioReset** function initializes (opens) the selected serial port. **SioReset** should be called before making any other calls to WSC except for setting default behavior (port=-1). **SioReset** uses the parity, stop bits, and word length value previously set if **SioParms** was called otherwise the default values (19200, no parity, 8 data, 1 stop) are used.

SioReset can be called with Port set to -1 in order to specify the behavior of DTR and RTS at port initialization:

```
SioReset(-1, DTR_Default, RTS_Default)
```

DTR will be set at port initialization if DTR_Default is 1, else DTR will be cleared. This is also the case for RTS_Default.

RETURNS

- `Return = WSC_IE_BADID` (No such port)
- `Return = WSC_IE_OPEN` (Already open)
- `Return = WSC_IE_MEMORY` (Cannot allocate memory)
- `Return = WSC_IE_HARDWARE` (Hardware error)

EXAMPLE

```
Code = SioReset(Port, 1024, 1024)
```

ALSO SEE

SioBaud, SioParms, and SioDone.

2.25 **SioRI** :: Reads the Ring Indicator (RI) modem status bit.

SYNTAX

```
SioRI(Port)

   Port : (I) Port selected.
```

REMARKS

The **SioRI** function is used to read the Ring Indicator (RI) modem status bit. It is recommended that incoming rings be detected by looking for the text "RING" in the input stream rather than the RI signal since some modems do not set the RI reliably.

RETURNS

- `Return = WSC_IE_NOPEN` (Port not opened. Call **SioReset** first)
- `Return = WSC_IE_BADID` (No such port)
- `Return = 0` (RI is clear)
- `Return = >0` (RI is set - RING has occurred)

EXAMPLE

```
Code = SioRI(Port)
```

ALSO SEE

SioRead.

2.26 **SioRTS** :: Sets, clears, or reads the Request to Send (RTS).

SYNTAX

```
SioRTS(Port, Cmd)

   Port : (I) Port selected.
   Cmd  : (I) RTS command (SET, CLEAR, or READ).
```

REMARKS

The **SioRTS** function controls the Request to Send (RTS bit in the modem control register).

The RTS line is used by some error correcting modems to implement hardware flow control. RTS is dropped by the computer to signal the modem not to send data and is raised to signal the modem to continue. RTS should be set when communicating with a modem unless flow control is being used.

Refer to the SERIAL User's Manual (SERIAL.PDF or http://www.marshallsoft.com/serial.htm) for a discussion of flow control. Commands (defined in WSC declaration file [Section 1.3]) are:

```
[NAME]          :  [FUNCTION]
WSC_SET_LINE    :  set RTS (ON)
WSC_CLEAR_LINE  :  clear RTS (OFF)
WSC_READ_LINE   :  read RTS
```

RETURNS

- `Return = WSC_IE_NOPEN` (Port not opened. Call **SioReset** first)
- `Return = WSC_IE_BADID` (No such port)
- `Return = WSC_RANGE` (Command is not one of 'S', 'C', or 'R')
- `Return = 0` (RTS is clear ['R' command])
- `Return > 0` (RTS is set ['R' command])

EXAMPLE

```
Code = SioRTS(Port, WSC_CLEAR_LINE)
```

ALSO SEE

SioFlow and SioDTR.

2.27 **SioRxClear** :: Clears the receive buffer.

SYNTAX

```
SioRxClear(Port)

   Port : (I) Port selected.
```

REMARKS

The **SioRxClear** function will delete any characters in the receive buffer (not the UART) for the specified port. After execution, the receive buffer will be empty.

RETURNS

- `Return = WSC_IE_NOPEN` (Port not opened. Call **SioReset** first)
- `Return = WSC_IE_BADID` (No such port)

EXAMPLE

```
     Code = SioRxClear(Port)
```

ALSO SEE

SioRxQue.

2.28 **SioRxQue** :: Returns the number of bytes in the receive queue.

SYNTAX

```
SioRxQue(Port)

   Port : (I) Port selected.
```

REMARKS

The **SioRxQue** function will return the number of bytes in the receive queue (not the UART) at the time of the call.

RETURNS

- `Return = WSC_IE_NOPEN` (Port not opened. Call **SioReset** first)
- `Return = WSC_IE_BADID` (No such port)

EXAMPLE

```
     Code = SioRxQue(Port)
```

ALSO SEE

See SioTxQue

2.29 **SioSetInteger** :: Sets integer parameter for serial processing.

SYNTAX

```
SioSetInteger(Port, ParamName, ParamValue)

Port      : (I) Port selected.
ParmName  : (I) Parameter name (integer code)
ParmValue : (L) Parameter value
```

REMARKS

The parameter values defined are as follows:

```
[NAME]           :  [FUNCTION]
WSC_WAIT_ON_PUTS :  Complete I/O before returning   ['W']
WSC_SIGNAL       :  Signal thread blocking on SioEvent ['S']
WSC_OVERLAPPED   :  Force overlapped I/O ['O']
```

WSC_WAIT_ON_PUTS is used to direct **SioPuts** to return immediately (before the I/O is complete) if ParamValue is TRUE (not 0). The default is 0 (FALSE), which means that **SioPuts** will not return until the I/O is completed.

WSC_SIGNAL is used to signal WSC to release the block created when **SioEvent** was called.

WSC_OVERLAPPED is used to disable all overlapped I/O (pass ParmValue = 0). By default, WSC32 will use overlapped I/O when running on Win98 (and above) machines, but not on Win95 machines, since Win95 does not support overlapped I/O.

RETURNS

The parameter value is returned if the parameter name is recognized, otherwise -1 is returned.

EXAMPLE

```
SioSetInteger(Port, WSC_WAIT_ON_PUTS, 1)

SioSetInteger(Port, WSC_SIGNAL, 1)
```

2.30 **SioSetTimeouts** :: Sets Transmit and Receive Timeout Constants.

SYNTAX

```
SioSetTimeouts(Port,ReadInter,ReadMult,ReadCons,WriteMult,WriteCons)

Port      : (I) port selected
ReadInter : (I) read interval t/o
ReadMult  : (I) read t/o multiplier
ReadCons  : (I) read t/o constant
WriteMult : (I) write t/o multiplier
WriteCons : (I) write t/o constant
```

REMARKS

Sets the transmit (**SioPutc**/**SioPuts**) & receive (**SioGetc**/**SioGets**) operation timeouts.

If the value returned by **SioPutc** is 0, then a timeout has occurred. If the value returned by **SioPuts** is less than the number of bytes passed as the last argument to **SioPuts**, then a timeout has occurred.

WSC_READ_INTERVAL_TIMEOUT (ReadInter)
Sets the maximum period of time (in milliseconds) allowed between two sequential bytes being read from the serial port before the receive operation terminates.

If set to MAXDWORD and the other two above READ timeouts are set to zero, then serial receive calls return immediately without waiting.

WSC_READ_TIMEOUT_MULTIPLIER (ReadMult)
Sets the multiplier (in milliseconds) used to calculate the overall timeout of serial receive operations. This timeout is given by:

```
NbrBytes * ReadTimeoutMultiplier + ReadTimeoutConstant   (NbrBytes = # bytes requested)
```

WSC_READ_TIMEOUT_CONSTANT (ReadConst)
Sets the constant (in milliseconds) used to calculate the overall timeout of serial receive operations. This timeout is given by:

```
NbrBytes * ReadTimeoutMultiplier + ReadTimeoutConstant   (NbrBytes = # bytes requested
```

WSC_WRITE_TIMEOUT_MULTIPLIER (WriteMult)
Sets the multiplier (in milliseconds) used to calculate the overall timeout of serial transmit operations. This timeout is given by:

```
NbrBytes * WriteTimeoutMultiplier + WriteTimeoutConstant   (NbrBytes = # bytes requested)
```

WSC_WRITE_TIMEOUT_CONSTANT (WriteCons)
Sets the constant (in milliseconds) used to calculate the overall timeout of serial transmit operations. This timeout is given by:

```
NbrBytes * WriteTimeoutMultiplier + WriteTimeoutConstant   (NbrBytes = # bytes requested)
```

RETURNS

- Return = WSC_WIN32ERR (cannot set timeouts)

EXAMPLE

```
SioSetTimeouts(Port,(DWORD)-1,(DWORD)0,(DWORD)0,(DWORD)1,(DWORD)2000);
```

2.31 **SioShortToByte** :: Converts 16-bit Unicode ASCII character buffer to 8-bit

SYNTAX

```
SioShortToByte(Buffer)

  Buffer : (P) character buffer
```

REMARKS

The **SioShortToByte** function converts the (null terminated) character buffer 'Buffer' from 16-bit Unicode ASCII characters to 8-bit ASCII characters.

The buffer must be null terminated (last character is a hex 00).

This function is only necessary when working with 16-bit Unicode ASCII characters.

RETURNS

None.

EXAMPLE (C#)

```
NameString = "MyFile.zip\0"
char[] NameBuffer = NameString.ToCharArray();
// convert (null terminated) 16-unicode buffer to 8-bit
fixed (char* pNameBuffer = NameBuffer)
SioShortToByte(pNameBuffer);
```

ALSO SEE

SioByteToShort

2.32 **SioStatus** :: Returns the serial port status.

SYNTAX

```
SioStatus(Port, Mask)

   Port : (I) Port selected.
   Mask : (I) Error mask.
```

REMARKS

The **SioStatus** function returns the serial port error status corresponding to the mask argument.

[MASK NAME]	:	[FUNCTION]
WSC_RXOVER	:	The receive queue overflowed.
WSC_OVERRUN	:	An incoming byte was overwritten.
WSC_PARITY	:	A parity error was detected (incoming byte)
WSC_FRAME	:	A framing error was detected (incoming byte)
WSC_BREAK	:	A break signal was detected.
WSC_TXFULL	:	The transmit queue is full.

RETURNS

- `Return = WSC_IE_NOPEN` (Port not opened. Call **SioReset** first)
- `Return = WSC_IE_BADID` (No such port)

EXAMPLE

```
Code = SioStatus(Port, WSC_FRAME)
```

ALSO SEE

SioRead.

2.33 **SioTimer** :: Returns the current time in milliseconds.

SYNTAX

```
SioTimer()
```

REMARKS

The **SioTimer** returns the system time in milliseconds. **SioTimer** calls the Windows API function GetCurrentTime. This function is provided as a convenience since GetCurrentTime could be called directly from most computer languages.

RETURNS

The system time in milliseconds.

EXAMPLE

```
TimeNow = SioTimer()
```

2.34 **SioTxClear** :: Clears the transmit buffer.

SYNTAX

```
SioTxClear(Port)

   Port : (I) Port selected.
```

REMARKS

The **SioTxClear** function will delete any characters in the transmit buffer (not the UART) for the specified port.

Once this function is called, any character in the transmit buffer (put there by **SioPutc** or **SioPuts**) will be lost and therefore not transmitted.

RETURNS

- `Return = WSC_IE_NOPEN` (Port not opened. Call **SioReset** first)
- `Return = WSC_IE_BADID` (No such port)

EXAMPLE

```
Code = SioTxClear(Port)
```

ALSO SEE

SioTxQue.

2.35 **SioTxQue** :: Returns the number of bytes in the transmit queue.

SYNTAX

```
SioTxQue(Port)

   Port : (I) Port selected.
```

REMARKS

The **SioTxQue** function will return the number of characters in the transmit queue (not the UART) at the time of the call.

RETURNS

- `Return = WSC_IE_NOPEN` (Port not opened. Call **SioReset** first)
- `Return = WSC_IE_BADID` (No such port)

EXAMPLE

```
     Code = SioTxQue(Port)
```

ALSO SEE

SioRxQue.

2.36 **SioUnGetc** :: "Ungets" the last character read with SioGetc().

SYNTAX

```
SioUnGetc(Port, Ch)

   Port : (I) Port selected.
   Ch   : (I) Character to unget.
```

REMARKS

The **SioUnGetc** function returns ("pushes") the character back into the serial input buffer. The character pushed will be the next character returned by **SioGetc**. Only one character can be pushed back. This function works just like the "ungetc" function in the C language.

RETURNS

- `Return = WSC_IE_NOPEN` (Port not opened. Call **SioReset** first)
- `Return = WSC_IE_BADID` (No such port)

EXAMPLE

```
     Code = SioUnGetc(Port)
```

ALSO SEE

SioReset.

2.37 **SioWinError** :: Return last Win32 error code & message text.

SYNTAX

```
SioWinError(Buffer, Size)

   Buffer : (P) Pointer to messages buffer.
   Size   : (I) Size of buffer.
```

REMARKS

The **SioWinError** is a Win32 ONLY function that returns the last Win32 error code. If 'Buffer' is not NULL, it will also copy the corresponding text message into 'Buffer' of maximum size 'Size'

EXAMPLE

C/C++ Example

```
     char Buffer[128]
     Code = SioWinError((LPSTR)Buffer, 128)
```

BASIC Example

```
     Dim Buffer As String * 128
     Code = SioWinError(Buffer, 128)
```

RETURNS

The Win32 numeric error code.

3 Modem I/O Functions

3.1 **mioBreak** :: Aborts the Modem I/O state driver.

SYNTAX

```
mioBreak(Port)

   Port : (I) Port selected.
```

REMARKS

Forces the MIO driver to the IDLE state, abandoning any work in progress (if any). Used to abort **mioSendTo**, **mioQuiet**, and **mioWaitFor** functions.

RETURNS

`Return =` MIO_IDLE.

EXAMPLE

```
     Code = mioBreak(Port)
```

3.2 **mioDriver** :: Modem I/O state driver.

SYNTAX

```
mioDriver(Port)

   Port : (I) Port selected.
```

REMARKS

Executes the next state of any previously started MIO function such as **mioSendTo, mioWaitFor**, and **mioQuiet**. Returns MIO_IDLE (defined in MIO.H) if idle (not running), MIO_RUNNING if running, and anything else that is a character from the modem that can be displayed if wanted.

RETURNS

- `Return = MIO_IDLE` (if the driver is ready for the next **mioSendTo**, **mioWaitFor**, or **mioQuiet**)
- `Return = MIO_RUNNING` (if the driver is not idle)
- `Return = <else>` (if the driver is not idle, and the returned character was received from the modem)

EXAMPLE

```
Code = mioDriver(Port)
```

3.3 **mioQuiet** :: Waits for Modem I/O state driver.

SYNTAX

```
mioQuiet(Port, Wait)

   Port : (I) Port selected.
   Wait : (L) Wait in milliseconds.
```

REMARKS

The **mioQuiet** function waits for continuous quiet [no incoming serial data] of 'Wait' milliseconds before returning. Any incoming characters while **mioQuiet** is running are lost.

RETURNS

```
Return = TRUE.
```

EXAMPLE

```
     Code = mioQuiet(Port, 1000)
```

3.4 **mioResult** :: Returns result of last mioWaitFor.

SYNTAX

```
mioResult(Port)

   Port : (I) Port selected.
```

REMARKS

The **mioResult** function returns the result of the last **mioWaitFor** function. This function should not be called until the driver returns MIO_IDLE. See the remarks section of the **mioWaitFor** function for an example.

RETURNS

- `Return = 0` (False - last WaitFor not matched)
- `Return = !0` ('0' if first substring matched, '1' if second substring matched, etc.)

EXAMPLE

```
     Code = mioResult(Port)
```

ALSO SEE

mioWaitFor.

3.5 **mioSendTo** :: Sends string to modem.

SYNTAX

```
mioSendTo(Port, Pace, Text)

   Port   : (I) Port selected.
   Pace   : (L) The inter-character delay in milliseconds.
   String : (P) The string to send.
```

REMARKS

The **mioSendTo** function sends the characters in the string 'Text' to serial output. There is a delay of 'Pace' milliseconds between characters. Three characters in 'Text' are interpreted as:

```
     [NAME]       :  [FUNCTION]
     char is '^'  :  next character is control char (^A for 0x01)
     char is '!'  :  replaced with carriage return.
     char is '~'  :  removed from string (delay 1/2 second).
```

RETURNS

```
Return = TRUE.
```

EXAMPLE

```
     Code = mioSendTo(Port, 100, "ATDT555~1212!")
```

3.6 **mioWaitFor** :: Waits for continuous quiet.

SYNTAX

```
mioWaitFor(Port, Wait, Text)

   Port : (I) Port selected.
   Wait : (L) Total time to wait for response (milliseconds).
   Text : (P) The expected response string.
```

REMARKS

The **mioWaitFor** function waits for characters from serial input that match the string 'Text'. A total of 'Wait' milliseconds are allowed before timing out and returning FALSE (0). The string comparison is NOT case sensitive.

The function **mioDriver**() must be called until MIO_IDLE is returned. Then mioResult() is called to get the result of the **mioWaitFor**. Looking at the example below, a value of 0 indicates that neither "CONNECT", "BUSY", nor "NO CARRIER" was received. A non-zero value indicates that one of the three sub-strings was received. A '0' is returned if "CONNECT" was seen, '1' is returned if "NO CARRIER" was seen, and '2' is returned if "BUSY" was seen.

RETURNS

A character as described above.

EXAMPLE

```
     Code = mioWaitFor(Port, 60000, "CONNECT|NO  CARRIER|BUSY")
```

ALSO SEE

mioResult.

4 XYM Functions

4.1 xyAbort :: Aborts the XYDRIVER state driver.

SYNTAX

```
xyAbort(Port)

   Port : (I) Port selected.
```

REMARKS

The **xyAbort** function forces the driver to IDLE, terminating any file transfer that may be in progress.

RETURNS

```
Return = XY_NO_ERROR (0).
```

EXAMPLE

```
     Code = xyAbort(Port)
```

4.2 **xyAcquire** :: Prepares the state driver for operation.

SYNTAX

```
xyAcquire(FirstPort, LastPort)

   FirstPort : (I) First port selected.
   LastPort  : (I) Last port selected.
```

REMARKS

The **xyAcquire** function initializes the driver for subsequent use. This should be the first driver function called.

RETURNS

- `Return = =0` (No error [XY_NO_ERROR])
- `Return = <0` (XYDRIVER error. See "XYDRIVER Error Codes")

EXAMPLE

```
     Code = xyAcquire(COM1, COM1)
```

ALSO SEE

xyRelease.

4.3 **xyDebug** :: Set the driver debug level.

SYNTAX

```
xyDebug(Level)

   Level : (I) Debug level value.
```

REMARKS

The **xyDebug** functions sets the driver debug level as follows:

```
[LEVEL]        [FUNCTION]
Level is 0  :  Only error messages are generated (default).
Level is 1  :  Minimal debug messages are generated.
Level is 2  :  Maximal debug messages are generated.
```

Debug messages are retrieved using the **xyGetMessage** function.

RETURNS

New debug level [0,1,2]

EXAMPLE

```
Code = xyDebug(0)
```

ALSO SEE

xyGetMessage.

4.4 **xyDriver** :: XMODEM / YMODEM state driver.

SYNTAX

```
xyDriver(Port)

   Port : (I) Port selected.
```

RETURNS

- `Return = XY_IDLE : A transfer is not underway.`

REMARKS

The **xyDriver** function drives the state engine. Note that xyDriver never returns an error code.

In order to send or to receive a file, call **xyDriver** in a loop until it returns XY_IDLE (numerical 0) after first initiating the transfer by calling either **xyStartTx** or **xyStartRx**.

xyDriver can be called as often as wanted whether or not a file transfer was initiated.

EXAMPLE

```
Code = xyDriver(Port)
```

ALSO SEE

xyStartTx and xyStartRx.

4.5 xyGetFileName :: Get the filename from packet 0

SYNTAX

```
xyGetFileName(Port, Buffer, Size)

   Port   : (I) Port selected.
   Buffer : (P) Filename buffer.
   Size   : (I) Size of Filename buffer.
```

REMARKS

The **xyGetFileName** function gets the current filename. This function is designed for use on the receive side YMODEM protocol, where the filename is received as part of the first packet (packet #0). See the TERM example program.

RETURNS

- `Return = TRUE` (A message was copied into Buffer)
- `Return = FALSE` (No messages are available)

EXAMPLE

`C/C++ Example`

```
char Buffer[40]
Code = xyGetFileName(Port, (LPSTR)Buffer, 40)
```

`BASIC Example`

```
Dim Buffer As String * 40
Code = xyGetFileName(Port, Buffer, 40)
```

ALSO SEE

xyGetParameter.

4.6 **xyGetMessage** :: Get next XYDRIVER message.

SYNTAX

```
xyGetMessage(Port, Buffer, Size)

   Port   : (I) Port selected.
   Buffer : (P) Message buffer.
   Size   : (I) Size of message buffer.
```

REMARKS

The **xyGetMessage** function retrieves the next message from the driver message queue. Refer to the TERM example program for an example of using **xyGetMessage**.

RETURNS

- `Return = TRUE` (A message was copied into Buffer)
- `Return = FALSE` (No messages are available)

EXAMPLE

`C/C++ Example`

```
char Buffer[40]
Code = xyGetMessage (Port, (LPSTR)Buffer, 40)
```

`BASIC Example`

```
Dim Buffer As String * 40
Code = xyGetMessage(Port, Buffer, 40)
```

ALSO SEE

xyDebug.

4.7 **xyGetParameter** :: Retrieves driver parameter.

SYNTAX

```
xyGetParameter(Port, Parm)

   Port : (I) Port Selected.
   Parm : (I) Parameter to return.
```

REMARKS

The parameter value corresponding to the following table is returned.

```
[NAME]               :  [FUNCTION]
XY_GET_VERSION       :  Returns XYM version (a.b.c).
XY_GET_BUILD         :  Returns XYM build number.
XY_GET_ERROR_CODE    :  Driver error code (see XYM.H)
XY_GET_ERROR_STATE   :  Error state (if in error).
XY_GET_PACKET        :  Current packet number.
XY_GET_STATE         :  Current state (see XYDRIVER.C).
XY_GET_FILE_SIZE     :  File size.
XY_GET_NBR_NAKS      :  Get number of packets ACK'ed.
XY_GET_LAST_GET      :  Last incoming (serial) character.
XY_GET_LAST_PUT      :  Last outgoing (serial) character.
XY_GET_GET_COUNT     :  Number of incoming characters (bytes).
XY_GET_PUT_COUNT     :  Number of outgoing characters (bytes).
XY_GET_DRIVER_COUNT  :  Number times xyDriver() was called.
XY_GET_SHORT_PACKETS :  Get number of short packets (RX side only).
XY_GET_PACKETS_ACKED :  Get number of packets ACK'ed.
-1                   :  Cannot recognize parameter.
```

The **xyGetParameter** function can be used to check the state of the driver. For example:

(1) **xyGetParameter**(Port, XY_GET_STATE) returns XY_IDLE if idle.
(2) **xyGetParameter**(Port, XY_GET_ERROR_CODE) returns the driver error code if an error has occurred or XY_NO_ERROR (0) otherwise.

RETURNS

See above.

EXAMPLE

```
Code = xyGetParameter(Port, XY_GET_VERSION)
```

4.8 **xyRelease** :: Releases driver port.

SYNTAX

```
xyRelease()
```

REMARKS

The **xyRelease** function releases the ports that were previously acquired with **xyAcquire**. This function should be called before calling the WSC function **SioDone**.

RETURNS

- `Return = 0` (No error [XY_NO_ERROR])
- `Return = <0` (XYDRIVER error. See "XYDRIVER Error Codes")

EXAMPLE

```
Code = xyRelease()
```

ALSO SEE

xyAcquire.

4.9 **xySetParameter** :: Sets driver parameter.

SYNTAX

```
xySetParameter(Port, ParmName, ParmValue)

   Port      : (I) Port Selected.
   ParmName  : (I) Parameter Name.
   ParmValue : (L) Parameter Value.
```

REMARKS

The ParmValue corresponding to the following table is set.

```
     [NAME]                        :  [FUNCTION]
     ParmName = XY_SET_NAK_RATE    :  Sets the prompt delay (in seconds).
     ParmName = XY_SET_EOF_CHAR    :  Sets the XMODEM pad character.
     ParmName = XY_SET_ONE_SECOND  :  Sets the # milliseconds second.
```

The XY_SET_NAK_RATE parameter sets the delay (in seconds) between prompts that the receiver transmits to the sender to start the file transfer. The legal range is 1 to 10 seconds.

The XY_SET_EOF_CHAR parameter sets the pad character used by XMODEM in padding the last packet to 128 bytes. The normal value is control-Z (hex 1A).

The `XY_SET_ONE_SECOND` parameter (if set to less than 1000) is used to speed up the protocol by reducing waits. To reduce all time delays to half of their default value, use 500.

RETURNS

See above.

EXAMPLE

```
     Code = xySetParameter(Port, XY_SET_EOF_CHAR, 0)
```

4.10 **xySetString** :: Set Upload/Download Directory String.

SYNTAX

```
xySetString(Port, ParamName, ParamString)

   Port        : (I) Port to use.
   ParamName   : (I) Parameter name
   ParamString : (P) Pointer to parameter string
```

REMARKS

They location of the local upload/download directory can be specified by passing `XY_SET_FILES_DIR` as the `ParamName` and a pointer to the requested directory as `ParamString`.

If the local upload/download directory is not specified, then the current directory is the default location.

RETURNS

- Return > 0 (No error)
- Return = -1 (`ParamName` is not recognized)

EXAMPLE

`C/C++ Example`

```
Code = xySetString(Port, XY_SET_FILES_DIR, "C:\\WINDOWS\TEMP");
```

`BASIC Example`

```
Code = xySetString(Port, XY_SET_FILES_DIR, "C:\WINDOWS\TEMP")
```

ALSO SEE

None.

4.11 **xyStartRx** :: Start XMODEM or YMODEM receive.

SYNTAX

```
xyStartRx(Port, Filename, NCGchar, Batch)

   Port     : (I) Port to use.
   Filename : (P) File to receive (XMODEM only).
   NCGchar  : (I) NAK, 'C', or 'G'.
   Batch    : (I) YMODEM flag (T/F).
```

REMARKS

The **xyStartRx** starts the XMODEM or YMODEM file receive. Once started, calls to **xyDriver** are made to execute the next state (or states). The **xyStartTx** function returns immediately. The protocols supported and their parameters are as follows:

```
[Protocol]  :  [NCGchar]  [BatchFlag]
XMODEM      :     NAK      FALSE          (Standard XMODEM)
XMODEM/CRC  :    'C'       FALSE
XMODEM/1K   :    'C'       FALSE
YMODEM      :    'C'       TRUE           (Standard YMODEM)
```

RETURNS

- `Return = TRUE` (No error)
- `Return = FALSE` (Not started. Port not active)

EXAMPLE

C/C++ Example

```
Code = xyStartRx(Port, "MYFILE.ZIP", 'C', 1)
```

BASIC Example

```
Code = xyStartRx(Port, "MYFILE.ZIP", ASC("C"), 1)
```

ALSO SEE

xyStartTx and xyDriver.

4.12 **xyStartTx** :: Start XMODEM or YMODEM transmit.

SYNTAX

```
xyStartTx(Port, Filename, OneK, Batch)

   Port     : (I) Port to use.
   Filename : (P) File to send.
   OneK     : (I) Want 1K blocks (T/F).
   Batch    : (I) YMODEM flag (T/F).
```

REMARKS

The **xyStartTx** starts the XMODEM or YMODEM file send. Once started, calls to **xyDriver** are made to execute the next state (or states). The **xyStartTx** function returns immediately. The protocols supported and their parameters are as follows:

```
[Protocol]  :  [OneKflag]  [BatchFlag]
XMODEM      :    FALSE       FALSE       Standard XMODEM
XMODEM/CRC  :    FALSE       FALSE
XMODEM/1K   :    TRUE        FALSE
YMODEM      :    TRUE        TRUE        Standard YMODEM
```

RETURNS

- `Return = TRUE` (No error)
- `Return = FALSE` (Not started. Port not active)

EXAMPLE

```
Code = xyStartTx(Port, "MYFILE.ZIP", 0, 1)
```

ALSO SEE

xyStartRx and xyDriver.

5 Error Codes

5.1 WSC Error Codes

[NAME]	:	[FUNCTION]
WSC_ABORTED	:	The evaluation version of WSC corrupted.
WSC_BUFFERS	:	Cannot allocate memory for buffers.
WSC_EXPIRED	:	Evaluation version expired.
WSC_KEYCODE	:	Bad key code value.
WSC_NO_DATA	:	No incoming serial data is available.
WSC_RANGE	:	A parameter is out of range.
WSC_THREAD	:	Cannot start thread.
WSC_WIN32ERR	:	Win32 system error.
WSC_IE_BADID	:	No such port.
WSC_IE_BAUDRATE	:	Unsupported byte size.
WSC_IE_BYTESIZE	:	Unsupported byte size.
WSC_IE_DEFAULT	:	Error in default parameters
WSC_IE_HARDWARE	:	COM port hardware not present
WSC_IE_MEMORY	:	Cannot allocate memory.
WSC_IE_NOPEN	:	Port not opened. Call SioReset first.
WSC_IE_OPEN	:	Port already opened.
WSC_IO_ERROR	:	An event error has occurred.

The WSC_ABORTED error occurs in the evaluation version only if there is a problem displaying the software info screen.

The WSC_WIN32ERR error code is returned only for Win32 system errors. Call **SioWinError** to retrieve the error message.

5.2 XYDRIVER Error Codes

Error codes are always negative, except for "no error". Most of these error conditions rarely occur. Also note that XYDRIVER functions can return WSC errors. An error message is queued when an error occurs which can be retrieved with **xyGetMessage**.

[NAME]	:	[FUNCTION]
XY_NO_ERROR	:	No error.
XY_UNKNOWN_ERROR	:	Unknown error.
XY_ALREADY_ACTIVE_ERROR	:	Port already acquired.
XY_CANNOT_OPEN_ERROR	:	Cannot open specified file.
XY_EMPTY_FILE_ERROR	:	Specified file is empty.
XY_NO_STARTUP_CHAR_ERROR	:	Must specify NAK, 'C', or 'G'.
XY_NOT_NCG_ERROR	:	Expected NAK, 'C', or 'G'.
XY_DISK_READ_ERROR	:	Error reading disk.
XY_NO_EOT_ACK_ERROR	:	EOT was not ACK'ed.
XY_INTERNAL_ERROR	:	Internal error!
XY_CANCELLED_ERROR	:	Other side canceled.
XY_OUT_OF_SYNC_ERROR	:	Protocol has lost synchronization.
XY_RETRIES_ERROR	:	Packet retry limit was exceeded.
XY_BAD_PACKET_NBR_ERROR	:	Incorrect packet number.
XY_TIMED_OUT_ERROR	:	Timed out waiting for other side.
XY_NO_SUCH_FILE_ERROR	:	No such file.
XY_NOT_ACTIVE_ERROR	:	Port not acquired by xyAcquire.
XY_PORT_RANGE_ERROR	:	Port number out of range.

The latest versions of WSC are available on our web site at

http://www.marshallsoft.com/serial-communication-library.htm

[END]