

## Namespaces

Namespaces

Namespaces

Namespace	Description
<a href="#">SciComm</a>	

Send comments on this topic to [support@microglyph.com](mailto:support@microglyph.com)

(c) 2007 MicroGlyph Systems. All rights reserved.

## SciComm Namespace

Namespaces > SciComm

Syntax

C#

namespace SciComm  Types

All Types

Classes

Structures

Interfaces

Enumerat

**Ic  
on**

**Type**

**Description**



**SC**

SC Class - SciComm Serial Communications Interface  
Methods

Send comments on this topic to [support@microglyph.com](mailto:support@microglyph.com)

(c) 2007 MicroGlyph Systems. All rights reserved.

## SC Class

Namespaces > **SciComm** > **SC**

SC Class - SciComm Serial Communications Interface Methods

### Syntax

C#

`public class SC`

All Members	Constructors	Methods	Properties	Fields
<input checked="" type="checkbox"/> Public		<input checked="" type="checkbox"/> Instance		<input checked="" type="checkbox"/> Decla
<input checked="" type="checkbox"/> Protected		<input checked="" type="checkbox"/> Static		<input checked="" type="checkbox"/> Inher

Icon	Member	Description
	<b>SC()</b>	SC Class - constructor
	<b>COMCLS(Int32, Int32)</b>	Routine to close serial port.
	<b>COMCTL(Int32, Int32)</b>	Routine to set control lines.
	<b>COMDTM(Int32, Int32, Int32, Int32, Int32, Int32)</b>	Routine to get date and time.
	<b>COMENC(Int32, Byte[], Int32[], String)</b>	Routine to encode bytes, integers, or strings
	<b>COMERR(String, String, Int32)</b>	Routine to get error messages and error code.
	<b>COMFLS(Int32, Int32)</b>	Routine to flush serial port buffers.
	<b>COMOPN(Int32, Int32, Int32, Int32, Int32, Int32, Int32)</b>	Routine to open serial port.
	<b>COMREC(Int32, Int32, Int32, Int32, Byte[], Byte[], Int32)</b>	Routine to read bytes from serial port buffer.
	<b>COMSND(Int32, Int32, Int32, Int32, Byte[])</b>	Routine to write bytes to serial port buffer.
	<b>COMSTS(Int32, Int32, Int32, Int32, Int32, Int32, Int32, Int32, Int32, Int32)</b>	Routine to get serial port status.
	<b>COMSUS(Int32)</b>	Routine to sleep the executing thread.
	<b>COMTMR()</b>	Routine to get current time in milliseconds.

### Inheritance

#### Hierarchy

Object

└─ SC

Assembly: SCICOMM (Module: SCICOMM)

Send comments on this topic to [support@microglyph.com](mailto:support@microglyph.com)

(c) 2007 MicroGlyph Systems. All rights reserved.

## SC Constructor

[Namespaces](#) > [SciComm](#) > [SC](#) > [SC\(\)](#)



SC Class - constructor

### Syntax

C#

```
public SC () Assembly: SCICOMM (Module: SCICOMM)
```

Send comments on this topic to [support@microglyph.com](mailto:support@microglyph.com)

(c) 2007 MicroGlyph Systems. All rights reserved.

## COMCLS Method (code, port)

Namespaces > [SciComm](#) > [SC](#) > [COMCLS\(Int32, Int32\)](#)

Routine to close serial port.

### Syntax

C#

```
public static int COMCLS (  
    int code,  
    int port  
)
```

### Parameters

code (**Int32**)

Function code

- 1 - Purge receive/send buffers
- 2 - Purge receive,wait on send buffers

port (**Int32**)

Serial port ID

- 1 - COM1
- 2 - COM2
- 3 - COM3
- 4 - COM4
- 5 - COM5
- 6 - COM6
- 7 - COM7
- 8 - COM8
- 9 - COM9
- 10 - COM10
- 11 - COM11
- 12 - COM12

### Return Value

Status is returned as an int value.

- 0 - Normal return
- 1 - Invalid argument
- 2 - System exception
- 3 - SC Class not instantiated
- 4 - Port is closed
- 5 - Send thread failed to stop

Assembly: SCICOMM (Module: SCICOMM)

Send comments on this topic to [support@microglyph.com](mailto:support@microglyph.com)

(c) 2007 MicroGlyph Systems. All rights reserved.

## COMCTL Method (code, port)

Namespaces > [SciComm](#) > [SC](#) > [COMCTL\(Int32, Int32\)](#)

Routine to set control lines.

### Syntax

C#

```
public static int COMCTL (  
    int code,  
    int port  
)
```

### Parameters

code (**Int32**)

Function code

- 1 - Raise data-terminal-ready signal
- 2 - Raise request-to-send signal
- 3 - Raise line-break signal
- 4 - Drop data-terminal-ready signal
- 5 - Drop request-to-send signal
- 6 - Drop line-break signal

port (**Int32**)

Serial port ID

- 1 - COM1
- 2 - COM2
- 3 - COM3
- 4 - COM4
- 5 - COM5
- 6 - COM6
- 7 - COM7
- 8 - COM8
- 9 - COM9
- 10 - COM10
- 11 - COM11
- 12 - COM12

### Return Value

Status is returned as an int value.

- 0 - Normal return
- 1 - Invalid argument
- 2 - System exception
- 3 - SC Class not instantiated

- 4 - Port is closed
- 5 - Handshake active error

Assembly: SCICOMM (Module: SCICOMM)

Send comments on this topic to [support@microglyph.com](mailto:support@microglyph.com)

(c) 2007 MicroGlyph Systems. All rights reserved.

## COMDTM Method (month, day, year, hour, min, sec, msec)

Namespaces > SciComm > SC > COMDTM(Int32, Int32, Int32, Int32, Int32, Int32,

Int32)

Routine to get date and time.

### Syntax

C#

```
public static int COMDTM (  
    ref int month,  
    ref int day,  
    ref int year,  
    ref int hour,  
    ref int min,  
    ref int sec,  
    ref int msec  
)
```

### Parameters

month (**Int32**)

Month

day (**Int32**)

Day

year (**Int32**)

Year

hour (**Int32**)

Hour

min (**Int32**)

Minute

sec (**Int32**)

Second

msec (**Int32**)

Millisecond

### Return Value

Status is returned as an int value.

- 0 - Normal return
- 1 - System exception
- 2 - SC Class not instantiated

Assembly: SCICOMM (Module: SCICOMM)

Send comments on this topic to [support@microglyph.com](mailto:support@microglyph.com)

(c) 2007 MicroGlyph Systems. All rights reserved.

## COMENC Method (code, bytbuf, intbuf, svalue)

Namespaces > [SciComm](#) > [SC](#) > **COMENC(Int32, Byte[], Int32[], String)**

Routine to encode bytes, integers, or strings

### Syntax

C#

```
public static int COMENC (  
    int code,  
    byte[] bytbuf,  
    int[] intbuf,  
    ref string svalue  
)
```

### Parameters

code (**Int32**)

Type of encoding

- 1 - int to Byte
- 2 - Byte to int
- 3 - string to Byte (ASCII encoding)
- 4 - string to Byte (Unicode encoding)
- 5 - Byte to string (ASCII encoding)
- 6 - Byte to string (Unicode encoding)

bytbuf (**Byte**[])

Byte array(4 times intbuf size)

intbuf (**Int32**[])

Integer array

svalue (**String**)

String value

### Return Value

Status is returned as an int value.

- 0 - Normal return
- 1 - Invalid arguments
- 2 - System exception
- 3 - SC Class not instantiated

Assembly: SCICOMM (Module: SCICOMM)

Send comments on this topic to [support@microglyph.com](mailto:support@microglyph.com)

(c) 2007 MicroGlyph Systems. All rights reserved.

## COMERR Method (Istexc, Istrtn, Isterr)

Namespaces > [SciComm](#) > [SC](#) > COMERR(String, String, Int32)

Routine to get error messages and error code.

### [-] Syntax

C#

```
public static int COMERR (  
    ref string Istexc,  
    ref string Istrtn,  
    ref int Isterr  
)
```

### [-] Parameters

Istexc (**String**)

Last SciComm Exception

Istrtn (**String**)

Last SciComm routine in error

Isterr (**Int32**)

Last SciComm routine error code

### [-] Return Value

Status is returned as an int value.

- 0 - Normal return
- 1 - Invalid arguments
- 2 - SC Class not instantiated
- 3 - System exception

### [-] Remarks

Method clears current error messages and code.

Assembly: SCICOMM (Module: SCICOMM)

Send comments on this topic to [support@microglyph.com](mailto:support@microglyph.com)

(c) 2007 MicroGlyph Systems. All rights reserved.

## COMFLS Method (code, port)

Namespaces > [SciComm](#) > [SC](#) > [COMFLS\(Int32, Int32\)](#)

Routine to flush serial port buffers.

### Syntax

C#

```
public static int COMFLS (  
    int code,  
    int port  
)
```

### Parameters

code (**Int32**)

Function code

- 1 - Flush receive buffer
- 2 - Flush send buffers
- 3 - Flush receive and send buffers

port (**Int32**)

Serial port ID

- 1 - COM1
- 2 - COM2
- 3 - COM3
- 4 - COM4
- 5 - COM5
- 6 - COM6
- 7 - COM7
- 8 - COM8
- 9 - COM9
- 10 - COM10
- 11 - COM11
- 12 - COM12

### Return Value

Status is returned as an int value.

- 0 - Normal return
- 1 - Invalid argument
- 2 - System exception
- 3 - SC Class not instantiated
- 4 - Port is closed

Assembly: SCICOMM (Module: SCICOMM)

Send comments on this topic to [support@microglyph.com](mailto:support@microglyph.com)

(c) 2007 MicroGlyph Systems. All rights reserved.

## COMOPN Method (port, baud, parity, stpbts, datbts, flow, reclen, sndlen)

Namespaces > **SciComm** > **SC** > COMOPN(Int32, Int32, Int32, Int32, Int32, Int32, Int32, Int32)

Routine to open serial port.

### Syntax

C#

```
public static int COMOPN (  
    int port,  
    int baud,  
    int parity,  
    int stpbts,  
    int datbts,  
    int flow,  
    int reclen,  
    int sndlen
```

### Parameters

port (**Int32**)

Serial port ID

- 1 - COM1
- 2 - COM2
- 3 - COM3
- 4 - COM4
- 5 - COM5
- 6 - COM6
- 7 - COM7
- 8 - COM8
- 9 - COM9
- 10 - COM10
- 11 - COM11
- 12 - COM12

baud (**Int32**)

Baud rate

- 1 - 110
- 2 - 300
- 3 - 600
- 4 - 1200
- 5 - 2400
- 6 - 4800
- 7 - 9600

- 8 - 14400
- 9 - 19200
- 10 - 38400
- 11 - 56000
- 12 - 57600
- 13 - 115200

parity (**Int32**)

Parity marking

- 1 - Even
- 2 - Mark
- 3 - None
- 4 - Odd
- 5 - Space

stpbits (**Int32**)

Stop bits per byte

- 1 - 0
- 2 - 1
- 3 - 1.5
- 4 - 2

datbts (**Int32**)

Data bits per byte  
4 to 8

flow (**Int32**)

Flow control

- 1 - None
- 2 - RTS/CTS
- 3 - RTS/CTS + XON/XOFF
- 4 - XON/XOFF

reclen (**Int32**)

Receive buffer size  
4096 (default and minimum value)

sndlen (**Int32**)

Send buffer size  
4096 (default and minimum value)

## ▣ Return Value

Status is returned as an int value.

- 0 - Normal return
- 1 - Invalid argument

- 2 - System exception
- 3 - SC Class not instantiated
- 4 - Port is open
- 5 - Send thread failed to start

Assembly: SCICOMM (Module: SCICOMM)

Send comments on this topic to [support@microglyph.com](mailto:support@microglyph.com)

(c) 2007 MicroGlyph Systems. All rights reserved.

## COMREC Method (code, port, nbytes, tmeout, delbuf, inpbuf, nread)

Namespaces > SciComm > SC > COMREC(Int32, Int32, Int32, Int32, Byte[], Byte[], Int32)

Routine to read bytes from serial port buffer.

### Syntax

C#

```
public static int COMREC (  
    int code,  
    int port,  
    int nbytes,  
    int tmeout,  
    byte[] delbuf,  
    byte[] inpbuf,  
    ref int nread
```

### Parameters

code (**Int32**)

Function code

- 1 - Read
- 2 - Read + timeout
- 3 - Read + timeout + delimiter

port (**Int32**)

Serial port ID

- 1 - COM1
- 2 - COM2
- 3 - COM3
- 4 - COM4
- 5 - COM5
- 6 - COM6
- 7 - COM7
- 8 - COM8
- 9 - COM9
- 10 - COM10
- 11 - COM11
- 12 - COM12

nbytes (**Int32**)

Number bytes to read (maximum)

tmeout (**Int32**)

Time out (ms)

delbuf (**Byte[]**)

Delimiter array

inpbuf (**Byte[]**)

Array to receive read bytes

nread (**Int32**)

Number of bytes read

#### **Return Value**

Status is returned as an int value.

- 0 - Normal return
- 1 - Invalid argument
- 2 - System exception
- 3 - SC Class not instantiated
- 4 - Port is closed
- 5 - Serial port errors (see COMSTS)
- 6 - No delimiter match
- 7 - Read timed out

Assembly: SCICOMM (Module: SCICOMM)

Send comments on this topic to [support@microglyph.com](mailto:support@microglyph.com)

(c) 2007 MicroGlyph Systems. All rights reserved.

## COMSND Method (code, port, nbytes, tmeout, outbuf)

Namespaces > [SciComm](#) > [SC](#) > COMSND(Int32, Int32, Int32, Int32, Byte[])

Routine to write bytes to serial port buffer.

### Syntax

C#

```
public static int COMSND (  
    int code,  
    int port,  
    int nbytes,  
    int tmeout,  
    byte[] outbuf  
)
```

### Parameters

code (**Int32**)

Function code

- 1 - Write
- 2 - Write + time stamp
- 3 - Write + timeout
- 4 - Write + time stamp + timeout

port (**Int32**)

Serial port ID

- 1 - COM1
- 2 - COM2
- 3 - COM3
- 4 - COM4
- 5 - COM5
- 6 - COM6
- 7 - COM7
- 8 - COM8
- 9 - COM9
- 10 - COM10
- 11 - COM11
- 12 - COM12

nbytes (**Int32**)

Number bytes to write

tmeout (**Int32**)

Time out (ms)

outbuf (**Byte**[])

Output bytes

### Return Value

Status is returned as an int value.

- 0 - Normal return

- 1 - Invalid argument
- 2 - System exception
- 3 - SC Class not instantiated
- 4 - Port is closed
- 5 - Write timed out
- 6 - Memory allocation error
- 7 - Send thread failure

Assembly: SCICOMM (Module: SCICOMM)

Send comments on this topic to [support@microglyph.com](mailto:support@microglyph.com)

(c) 2007 MicroGlyph Systems. All rights reserved.

## COMSTS Method (port, lsreg, msreg, baudrate, parity, stpbts, datbts, flow, reccnt, reclen, sndcnt, sndlen)

Namespaces > SciComm > SC > COMSTS(Int32, Int32, Int32, Int32, Int32, Int32,

Int32, Int32, Int32, Int32, Int32, Int32)

Routine to get serial port status.

### Syntax

C#

```
public static int COMSTS (  
    int port,  
    ref int lsreg,  
    ref int msreg,  
    ref int baudrate,  
    ref int parity,  
    ref int stpbts,  
    ref int datbts,  
    ref int flow,  
    ref int reccnt,  
    ref int reclen,  
    ref int sndcnt,  
    ref int sndlen  
)
```

### Parameters

port (**Int32**)

Serial port ID

- 1 - COM1
- 2 - COM2
- 3 - COM3
- 4 - COM4
- 5 - COM5
- 6 - COM6
- 7 - COM7
- 8 - COM8
- 9 - COM9
- 10 - COM10
- 11 - COM11
- 12 - COM12

lsreg (**Int32**)

Line status register

- 7 - Not used
- 6 - Xmit shift register empty
- 5 - Xmit hold register empty
- 4 - Line break detected

- 3 - Framing error
- 2 - Parity error
- 1 - Overrun error
- 0 - Data ready

msreg (**Int32**)

Modem status register

- 7 - Carrier detect signal
- 6 - Ring indicate
- 5 - Data set ready
- 4 - Clear to send
- 3 - Not used
- 2 - Not used
- 1 - Not used
- 0 - Not used

baudrate (**Int32**)

Baud rate

parity (**Int32**)

Parity marking

- 1 - Even
- 2 - Mark
- 3 - None
- 4 - Odd
- 5 - Space

stpbits (**Int32**)

Stop bits per byte

- 1 - 0
- 2 - 1
- 3 - 1.5
- 4 - 2

datbts (**Int32**)

Data bits per byte  
4 to 8

flow (**Int32**)

Flow control

- 1 - None
- 2 - RTS/CTS
- 3 - RTS/CTS + XON/XOFF

•4 - XON/XOFF

recCnt (**Int32**)  
Number bytes in receive buffer  
recLen (**Int32**)  
Length of receive buffer  
sndCnt (**Int32**)  
Number bytes in send buffer  
sndLen (**Int32**)  
Length of send buffer

▣ **Return Value**

Status is returned as an int value.

- 0 - Normal return
- 1 - Invalid argument
- 2 - System exception
- 3 - SC Class not instantiated
- 4 - Port is closed

Assembly: SCICOMM (Module: SCICOMM)

Send comments on this topic to [support@microglyph.com](mailto:support@microglyph.com)

(c) 2007 MicroGlyph Systems. All rights reserved.

## COMSUS Method (tmeout)

Namespaces > [SciComm](#) > [SC](#) > [COMSUS\(Int32\)](#)

Routine to sleep the executing thread.

### Syntax

C#

```
public static int COMSUS (  
    int tmeout  
)
```

### Parameters

tmeout (**Int32**)

Time out value

- 0 - Give up time slice to any other
- xxx - Sleep current thread for xxx

### Return Value

Status is returned as an int value.

- 0 - Normal return
- 1 - Argument error
- 2 - System exception
- 3 - SC Class not instantiated

Assembly: SCICOMM (Module: SCICOMM)

Send comments on this topic to [support@microglyph.com](mailto:support@microglyph.com)

(c) 2007 MicroGlyph Systems. All rights reserved.

## COMTMR Method

[Namespaces](#) > [SciComm](#) > [SC](#) > **COMTMR()**

Routine to get current time in milliseconds.

### **Syntax**

C#

```
public static int COMTMR ()  Return Value
```

Current time (milliseconds) is returned as an int value.

Assembly: SCICOMM (Module: SCICOMM)

Send comments on this topic to [support@microglyph.com](mailto:support@microglyph.com)

(c) 2007 MicroGlyph Systems. All rights reserved.

## Example Programs

Example Programs are provided to demonstrate the usage of SciComm Library methods.

The following links provide access to the example programs:

[TCOM](#)

Use of SciComm Library Methods.

```

!*****
!
!   TCOM - Program to test SciComm Serial Communications Library   *
!   *
!*****

!   Main Control Program

        PROGRAM MAIN

! .nca (MAIN,add=System.STAThreadAttribute())

        USE System
        USE System.Text
        USE MicroGlyph.SciComm

        IMPLICIT NONE

!   Instantiate MicroGlyph.SciComm Class

        TYPE(SC)          :: SCClass
        TYPE(SC), POINTER :: SCObject
        ALLOCATE(SCObject, SOURCE=SC())

!   Local variables

        INTEGER(4) :: port, baud, parity, stpbts, datbts, flow, reflen, sndlen, &
&   lsreg, msreg, reccnt, sndcnt, lsterr, NBytes, tmeout, NRead, rtnsts, &
&   func, i, j, j1, j2, n, status, code, ival, baudrate
        LOGICAL(4) :: dialog
        UIINTEGER(KIND=1), POINTER, DIMENSION(:) :: TmpBuf
        UIINTEGER(KIND=1), POINTER, DIMENSION(:) :: BytBuf
        UIINTEGER(KIND=1), POINTER, DIMENSION(:) :: DelBuf
        INTEGER(4), POINTER, DIMENSION(:) :: IntBuf
        CHARACTER(1) :: reply
        CHARACTER(2) :: CRLF
        CHARACTER(5) :: ifmt, iline
        CHARACTER(5), DIMENSION(12) :: PrtNames = ("COM1 ", "COM2 ", &
&   "COM3 ", "COM4 ", "COM5 ", "COM6 ", "COM7 ", "COM8 ", "COM9 ", &
&   "COM10", "COM11", "COM12"/)
        CHARACTER(5), DIMENSION(5) :: ParNames = ("Even ", "Mark ", &
&   "None ", "Odd ", "Space"/)
        CHARACTER(12), DIMENSION(4) :: StpNames = ("None ", &
&   "One ", "OnePointFive", "Two "/)
        CHARACTER(18), DIMENSION(4) :: FloNames = ("None ", &
&   "Rts/Cts ", "Rts/Cts + Xon/Xoff", "Xon/Xoff "/)
        TYPE(System.Exception), POINTER :: se
        TYPE(String), POINTER :: line, pline

!   Instantiate arrays

        ALLOCATE(DelBuf(2))
        ALLOCATE(IntBuf(1024))
        CRLF = CHAR(13) // CHAR(10)
        ifmt = ' (I) '

!   Start Main

        TRY

```

```

! Dialog loop

dialog = .TRUE.
DO WHILE (dialog)

    CALL Console.WriteLine (USTRING (CRLF // " TCOM Functions:" &
&                                     // CRLF))
    CALL Console.WriteLine (USTRING (                                     &
&                                     " O = Open (COMOPN) ") &
&                                     CALL Console.WriteLine (USTRING (                                     &
&                                     " C = Close (COMCLS) ") &
&                                     CALL Console.WriteLine (USTRING (                                     &
&                                     " S = Status (COMSTS) ") &
&                                     CALL Console.WriteLine (USTRING (                                     &
&                                     " R = Read (COMREC) ") &
&                                     CALL Console.WriteLine (USTRING (                                     &
&                                     " W = Write (COMSND) ") &
&                                     CALL Console.WriteLine (USTRING (                                     &
&                                     " F = Flush (COMFLS) ") &
&                                     CALL Console.WriteLine (USTRING (                                     &
&                                     " X = Control (COMCTL) ") &
&                                     CALL Console.WriteLine (USTRING (                                     &
&                                     " M = Display Menu") &
&                                     CALL Console.WriteLine (USTRING (                                     &
&                                     " Q = Quit/Exit") &
&                                     CALL Console.Write (USTRING (                                     &
&                                     CRLF // " Enter Function: ") &
line => Console.ReadLine ()
reply = ASTRING (line)
ival = ICHAR (reply)
IF (ival.gt.96) ival = ival - 32
reply = CHAR (ival)

! Process reply

SELECT CASE (reply)

CASE (' ')

! Blank character

CASE ('O')

! Open serial port

! Get Serial Port ID
CALL Console.WriteLine (USTRING (CRLF // &
&                                     " Port:" // CRLF))
CALL Console.WriteLine (USTRING (" 1 = COM1"))
CALL Console.WriteLine (USTRING (" 2 = COM2"))
CALL Console.WriteLine (USTRING (" 3 = COM3"))
CALL Console.WriteLine (USTRING (" 4 = COM4"))
CALL Console.WriteLine (USTRING (" 5 = COM5"))
CALL Console.WriteLine (USTRING (" 6 = COM6"))
CALL Console.WriteLine (USTRING (" 7 = COM7"))
CALL Console.WriteLine (USTRING (" 8 = COM8"))
CALL Console.WriteLine (USTRING (" 9 = COM9"))
CALL Console.WriteLine (USTRING (" 10 = COM10"))
CALL Console.WriteLine (USTRING (" 11 = COM11"))

```

```

CALL Console.WriteLine(USSTRING("          12 = COM12"))
CALL Console.Write(USSTRING(CRLF // "          Enter Port[1]: "))
port = 1
line => Console.ReadLine()
IF (line.Length.GT.0) THEN
    iline = ASTRING(line)
    READ(iline,ifmt)ival
!       Set selected value
        port = ival
END IF
!
Set baud
CALL Console.WriteLine(USSTRING(CRLF // "          Baud Rate:" // &
&                                     CRLF))
CALL Console.WriteLine(USSTRING("          1 = 110"))
CALL Console.WriteLine(USSTRING("          2 = 300"))
CALL Console.WriteLine(USSTRING("          3 = 600"))
CALL Console.WriteLine(USSTRING("          4 = 1200"))
CALL Console.WriteLine(USSTRING("          5 = 2400"))
CALL Console.WriteLine(USSTRING("          6 = 4800"))
CALL Console.WriteLine(USSTRING("          7 = 9600"))
CALL Console.WriteLine(USSTRING("          8 = 14400"))
CALL Console.WriteLine(USSTRING("          9 = 19200"))
CALL Console.WriteLine(USSTRING("         10 = 38400"))
CALL Console.WriteLine(USSTRING("         11 = 56000"))
CALL Console.WriteLine(USSTRING("         12 = 57600"))
CALL Console.WriteLine(USSTRING("         13 = 115200"))
CALL Console.Write(USSTRING(CRLF // "          Enter Baud[7]: "))
baud = 7
line => Console.ReadLine()
IF (line.Length.GT.0) THEN
    iline = ASTRING(line)
    READ(iline,ifmt)ival
!       Set selected value
        baud = ival
END IF
!
Set parity
CALL Console.WriteLine(USSTRING(CRLF // "          Parity:" // &
&                                     CRLF))
CALL Console.WriteLine(USSTRING("          1 = Even"))
CALL Console.WriteLine(USSTRING("          2 = Mark"))
CALL Console.WriteLine(USSTRING("          3 = None"))
CALL Console.WriteLine(USSTRING("          4 = Odd"))
CALL Console.WriteLine(USSTRING("          5 = Space"))
CALL Console.Write(USSTRING(CRLF // "          Enter Parity[3]: "))
parity = 3
line  => Console.ReadLine()
IF (line.Length.GT.0) THEN
    iline = ASTRING(line)
    READ(iline,ifmt)ival
!       Set selected value
        parity = ival
END IF
!
Set stop bits
CALL Console.WriteLine(USSTRING(CRLF // "          Stop Bits:" // &
&                                     CRLF))
CALL Console.WriteLine(USSTRING("          1 = 0"))
CALL Console.WriteLine(USSTRING("          2 = 1"))
CALL Console.WriteLine(USSTRING("          3 = 1.5"))
CALL Console.WriteLine(USSTRING("          4 = 2"))
CALL Console.Write(USSTRING(CRLF // "          &

```

```

&          "      Enter Stop Bits[2]: ")
stpbts = 2
line  => Console.ReadLine()
IF (line.Length.GT.0) THEN
    iline = ASTRING(line)
    READ(iline,ifmt)ival
!      Set selected value
    stpbts = ival
END IF
!
Set data bits
CALL Console.WriteLine(USTRING(CRLF //
&          "      Data Bits:" // CRLF))
CALL Console.WriteLine(USTRING("          4 to 8"))
CALL Console.Write(USTRING(CRLF //
&          "      Enter Data Bits[8]: "))

datbts = 8
line  => Console.ReadLine()
IF (line.Length.GT.0) THEN
    iline = ASTRING(line)
    READ(iline,ifmt)ival
!      Set selected value
    datbts = ival
END IF
!
Set flow
CALL Console.WriteLine(USTRING(CRLF //
&          "      Flow:" // CRLF))
CALL Console.WriteLine(USTRING("          1 = None"))
CALL Console.WriteLine(USTRING("          2 = Rts/Cts"))
CALL Console.WriteLine(USTRING(
&          "          3 = Rts/Cts + Xon/Xoff"))
CALL Console.WriteLine(USTRING("          4 = Xon/Xoff"))
CALL Console.Write(USTRING(CRLF // "      Enter Flow[1]: "))
flow = 1
line => Console.ReadLine()
IF (line.Length.GT.0) THEN
    iline = ASTRING(line)
    READ(iline,ifmt)ival
!      Set selected value
    flow = ival
END IF
!
Set reclen
CALL Console.Write(USTRING("      Enter Reclen[4096]:"))
reclen = 4096
line  => Console.ReadLine()
IF (line.Length.GT.0) THEN
    iline = ASTRING(line)
    READ(iline,ifmt)ival
!      Set selected value
    reclen = ival
END IF
!
Set sndlen
CALL Console.Write(USTRING("      Enter Sndlen[4096]:"))
sndlen = 4096
line  => Console.ReadLine()
IF (line.Length.GT.0) THEN
    iline = ASTRING(line)
    READ(iline,ifmt)ival
!      Set selected value
    sndlen = ival
END IF

```

```

status = SC.COMOPN(port,baud,parity,stpbits,darbts,flow,      &
&          reclen,sndlen)
IF (status.NE.0) THEN
  CALL Console.WriteLine(USTRING(                                &
&          CRLF // "          COMOPN Errors:" // CRLF))
  CALL Console.WriteLine(USTRING("          Status = {0}"),      &
&          status)
  CALL Console.WriteLine(USTRING("          Port   = {0}"),      &
&          port)
  CALL Console.WriteLine(USTRING("          Baud   = {0}"),      &
&          baud)
  CALL Console.WriteLine(USTRING("          Parity = {0}"),      &
&          parity)
  CALL Console.WriteLine(USTRING("          Stpbits = {0}"),      &
&          stpbits)
  CALL Console.WriteLine(USTRING("          Darbts = {0}"),      &
&          darbts)
  CALL Console.WriteLine(USTRING("          Flow   = {0}"),      &
&          flow)
  CALL Console.WriteLine(USTRING("          Reclen = {0}"),      &
&          reclen)
  CALL Console.WriteLine(USTRING("          Sndlen = {0}"),      &
&          sndlen)
END IF

CASE ('C')

!   Close serial port

  CALL Console.WriteLine(USTRING(CRLF //                          &
&          "          Close Functions:" // CRLF))
  CALL Console.WriteLine(USTRING(                                &
&          "          1 = Purge Receive/Send Buffers"))
  CALL Console.WriteLine(USTRING(                                &
&          "          2 = Purge Receive/Wait on" //              &
&          "          Send Buffers"))
  CALL Console.WriteLine(USTRING(CRLF //                          &
&          "          Enter Function[1]: "))

  code = 1
  line => Console.ReadLine()
  IF (line.Length.GT.0) THEN
    iline = ASTRING(line)
    READ(iline,ifmt)ival
!    Set selected value
    code = ival
  END IF
  CALL Console.WriteLine(USTRING(                                &
&          "          Enter Port   [1]: "))

  port = 1
  line => Console.ReadLine()
  IF (line.Length.GT.0) THEN
    iline = ASTRING(line)
    READ(iline,ifmt)ival
!    Set selected value
    port = ival
  END IF
  status = SC.COMCLS(code,port)
  IF (status.NE.0) THEN
    CALL Console.WriteLine(USTRING(                                &
&          CRLF // "          COMCLS Errors:" // CRLF))

```

```

        CALL Console.WriteLine(USTRING("        Status = {0}"), &
&          status)
        CALL Console.WriteLine(USTRING("        Code   = {0}"), &
&          code)
        CALL Console.WriteLine(USTRING("        Port   = {0}"), &
&          port)
    END IF

CASE ('M')

!   Menu command

CASE ('S')

!   Serial port status

    CALL Console.Write(USTRING(CRLF //                                &
&          "        Enter Port   [1]: "))
    port = 1
    line => Console.ReadLine()
    IF (line.Length.GT.0) THEN
        iline = ASTRING(line)
        READ(iline,ifmt)ival
!       Set selected value
        port = ival
    END IF
    status = SC.COMSTS(port,lsreg,msreg,baudrate,parity,stpmts, &
&          datbts,flow,recnt,reclen,sndcnt,sndlen)
    IF (status.EQ.0) &
&      THEN
&        CALL Console.WriteLine(USTRING(CRLF //                                &
&          "        Port   = " // PrtNames(port)))
&        CALL Console.WriteLine(USTRING("        Lsreg  = {0}"), &
&          lsreg)
&        CALL Console.WriteLine(USTRING("        Msreg  = {0}"), &
&          msreg)
&        CALL Console.WriteLine(USTRING("        Baud   = {0}"), &
&          baudrate)
&        CALL Console.WriteLine(USTRING(                                &
&          "        Parity = " // ParNames(parity)))
&        CALL Console.WriteLine(USTRING(                                &
&          "        Stpmts = " // StpNames(stpmts)))
&        CALL Console.WriteLine(USTRING("        Datbts = {0}"), &
&          datbts)
&        CALL Console.WriteLine(USTRING(                                &
&          "        Flow   = " // FloNames(flow)))
&        CALL Console.WriteLine(USTRING("        Recnt  = {0}"), &
&          recnt)
&        CALL Console.WriteLine(USTRING("        Reclen = {0}"), &
&          reclen)
&        CALL Console.WriteLine(USTRING("        Sndcnt = {0}"), &
&          sndcnt)
&        CALL Console.WriteLine(USTRING("        Sndlen = {0}"), &
&          sndlen)
    ELSE
&        CALL Console.WriteLine(USTRING(                                &
&          CRLF // "        COMSTS Errors:" // CRLF))
&        CALL Console.WriteLine(USTRING("        Status = {0}"), &
&          status)
&        CALL Console.WriteLine(USTRING("        Port   = {0}"), &

```

```

&                                     port)
    END IF

CASE ('R')

!   Read bytes from receive buffer

    CALL Console.WriteLine(USTRING(CRLF //
&                                     "    Receive Functions:" // CRLF)) &
    CALL Console.WriteLine(USTRING("    1 = Read"))
    CALL Console.WriteLine(USTRING(
&                                     "    2 = Read with Timeout")) &
    CALL Console.WriteLine(USTRING(
&                                     "    3 = Read with Timeout/Delimiter")) &
    CALL Console.Write(USTRING(CRLF //
&                                     "    Enter Function[1]: "))

    code = 1
    line => Console.ReadLine()
    IF (line.Length.GT.0) THEN
        iline = ASTRING(line)
        READ(iline,ifmt)ival
!       Set selected value
        code = ival
    END IF
    CALL Console.Write(USTRING(
&                                     "    Enter Port    [1]: "))

    port = 1
    line => Console.ReadLine()
    IF (line.Length.GT.0) THEN
        iline = ASTRING(line)
        READ(iline,ifmt)ival
!       Set selected value
        port = ival
    END IF
    tmeout    = 3000
    DelBuf(1) = 13
    DelBuf(2) = 10
    NBytes    = 4096
    ALLOCATE(TmpBuf(NBytes))
    status    = SC.COMREC(code,port,NBytes,tmeout,DelBuf,
&                                     TmpBuf,NRead) &
    IF (status.EQ.0)
&
&       THEN
!       Display bytes received
        IF (NRead.EQ.0) THEN
&           CALL Console.WriteLine(USTRING(
&                                     CRLF // "    No data available "
&                                     // CRLF))
&
&           CALL Console.WriteLine(USTRING(
&                                     "    Code    = {0}"),code) &
&           CALL Console.WriteLine(USTRING(
&                                     "    Port    = {0}"),port) &
        END IF
        IF (NRead.GT.0) THEN
            NBytes = NRead
            ALLOCATE(BytBuf(NBytes))
!           Move number of bytes read into Buffer
            DO i = 1, NRead
                BytBuf(i) = TmpBuf(i)
            END DO

```

```

CALL Console.WriteLine(USSTRING(CRLF //
!           "           Count: {0}"),NRead)           &
&
Convert bytes read to ASCII string
code      = 5
rtnsts = SC.COMENC(code,BytBuf,IntBuf,line)
! Display ASCII string
DO i = 1 , NBytes, 32
  n = Min(32,NBytes-i+1)
  pline => line.Substring(i-1,n)
  IF (i.EQ.1)
&           THEN
&           CALL Console.WriteLine(USSTRING(
&           "           ASCII: {0}"),pline)
&           ELSE
&           CALL Console.WriteLine(USSTRING(
&           "           : {0}"),pline)
&           END IF
END DO
! Display Hex string
DO i = 1 , NBytes, 11
  j1 = i
  j2 = i + Min(11,NBytes-i+1) - 1
  IF (i.EQ.1)
&           THEN
&           CALL Console.Write(USSTRING("           Hex : "))
&           DO j = j1 , j2
&           CALL Console.Write(USSTRING(
&           "{0,2:X2} "),BytBuf(j))
&           END DO
&           CALL Console.WriteLine(USSTRING(" "))
&           ELSE
&           CALL Console.Write(USSTRING("           : "))
&           DO j = j1 , j2
&           CALL Console.Write(USSTRING(
&           "{0,2:X2} "),BytBuf(j))
&           END DO
&           CALL Console.WriteLine(USSTRING(" "))
&           END IF
END DO
END IF
ELSE
& CALL Console.WriteLine(USSTRING(
& CRLF // "           COMREC Errors:" // CRLF))
& CALL Console.WriteLine(USSTRING("           Status = {0}"),
& status)
& CALL Console.WriteLine(USSTRING("           Code = {0}"),
& code)
& CALL Console.WriteLine(USSTRING("           Port = {0}"),
& port)
END IF

CASE ('W')
! Write out bytes to send buffer

CALL Console.WriteLine(USSTRING(CRLF //
&           "           Write Functions:" // CRLF))
& CALL Console.WriteLine(USSTRING("           1 = Write"))
& CALL Console.WriteLine(USSTRING(
&           "           2 = Write with Timestamp"))

```

```

CALL Console.WriteLine(USTRING(                                     &
&     "          3 = Write with Timeout"))                          &
CALL Console.WriteLine(USTRING(                                     &
&     "          4 = Write with Timeout/Timestamp"))                &
CALL Console.Write(USTRING(CRLF //                                 &
&     "          Enter Function[1]: "))
code = 1
line => Console.ReadLine()
IF (line.Length.GT.0) THEN
    iline = ASTRING(line)
    READ(iline,ifmt)ival
!     Set selected value
    code = ival
END IF
CALL Console.Write(USTRING(                                       &
&     "          Enter Port      [1]: "))
port = 1
line => Console.ReadLine()
IF (line.Length.GT.0) THEN
    iline = ASTRING(line)
    READ(iline,ifmt)ival
!     Set selected value
    port = ival
END IF
CALL Console.Write(USTRING("          Enter String      : "))
line  => Console.ReadLine()
NBytes = line.Length
tmeout = 3000
! Convert string to ASCII bytes
func   = 3
ALLOCATE(BytBuf(NBytes))
status = SC.COMENC(func,BytBuf,IntBuf,line)
! Send the ASCII bytes
status = SC.COMSND(code,port,NBytes,tmeout,BytBuf)
IF (status.NE.0) THEN
    CALL Console.WriteLine(USTRING(                                     &
&         CRLF // "          COMSND Errors:" // CRLF))                &
    CALL Console.WriteLine(USTRING("          Status = {0}"), &
&         status)
    CALL Console.WriteLine(USTRING("          Code   = {0}"), &
&         code)
    CALL Console.WriteLine(USTRING("          Port   = {0}"), &
&         port)
END IF

CASE ('F')
! Flush receive and send buffers

CALL Console.WriteLine(USTRING(CRLF //                                 &
&     "          Flush Functions:" // CRLF))                          &
CALL Console.WriteLine(USTRING(                                     &
&     "          1 = Flush Receive Buffers"))                          &
CALL Console.WriteLine(USTRING(                                     &
&     "          2 = Flush Send Buffers"))                              &
CALL Console.WriteLine(USTRING(                                     &
&     "          3 = Flush Receive/Send Buffers"))                      &
CALL Console.Write(USTRING(CRLF //                                 &
&     "          Enter Function[1]: "))
code = 1

```

```

line => Console.ReadLine()
IF (line.Length.GT.0) THEN
    iline = ASTRING(line)
    READ(iline,ifmt)ival
!    Set selected value
    code = ival
END IF
CALL Console.Write(USTRING(
&
&        "    Enter Port    [1]: "))
port = 1
line => Console.ReadLine()
IF (line.Length.GT.0) THEN
    iline = ASTRING(line)
    READ(iline,ifmt)ival
!    Set selected value
    port = ival
END IF
status = SC.COMFLS(code,port)
IF (status.NE.0) THEN
    CALL Console.WriteLine(USTRING(
&
&        CRLF // "    COMFLS Errors:" // CRLF))
    CALL Console.WriteLine(USTRING("    Status = {0}"),
&
&        status)
    CALL Console.WriteLine(USTRING("    Code   = {0}"),
&
&        code)
    CALL Console.WriteLine(USTRING("    Port   = {0}"),
&
&        port)
END IF

CASE ('X')
!
    Set control lines

    CALL Console.WriteLine(USTRING(CRLF //
&
&        "    Control Functions:" // CRLF))
    CALL Console.WriteLine(USTRING(
&
&        "    1 = Raise Data-Terminal-Ready Signal"))
    CALL Console.WriteLine(USTRING(
&
&        "    2 = Raise Request-to-Send Signal"))
    CALL Console.WriteLine(USTRING(
&
&        "    3 = Raise Line-Break Signal"))
    CALL Console.WriteLine(USTRING(
&
&        "    4 = Drop Data-Terminal-Ready Signal"))
    CALL Console.WriteLine(USTRING(
&
&        "    5 = Drop Request-to-Send Signal"))
    CALL Console.WriteLine(USTRING(
&
&        "    6 = Drop Line-Break Signal"))
    CALL Console.Write(USTRING(CRLF //
&
&        "    Enter Function[1]: "))
code = 1
line => Console.ReadLine()
IF (line.Length.GT.0) THEN
    iline = ASTRING(line)
    READ(iline,ifmt)ival
!    Set selected value
    code = ival
END IF
CALL Console.Write(USTRING(
&
&        "    Enter Port    [1]: "))
port = 1

```

```

        line => Console.ReadLine()
        IF (line.Length.GT.0) THEN
            iline = ASTRING(line)
            READ(iline,ifmt)ival
!           Set selected value
            port = ival
        END IF
        status = SC.COMCTL(code,port)
        IF (status.NE.0) THEN
            CALL Console.WriteLine(USTRING(
&                CRLF // "          COMCTL Errors:" // CRLF)) &
            CALL Console.WriteLine(USTRING("          Status = {0}"), &
&                status) &
            CALL Console.WriteLine(USTRING("          Code   = {0}"), &
&                code) &
            CALL Console.WriteLine(USTRING("          Port   = {0}"), &
&                port)
        END IF

        CASE ('Q')
!           Exit processing

            CALL Console.WriteLine(USTRING(" "))
            dialog = .FALSE.

        CASE DEFAULT
!           Unknown command

        END SELECT

    END DO

    STOP

    CATCH (se)
!           Report exception in Main Program

            CALL Console.WriteLine(USTRING(
&                " Main Program Exception {0}"),se.Message) &

    END TRY

    END PROGRAM

```