

Namespaces

Namespaces

Namespaces

Namespace	Description
SciSnet	

Send comments on this topic to support@microglyph.com

(c) 2007 MicroGlyph Systems. All rights reserved.

SciSnet Namespace

Namespaces > SciSnet

Syntax

J#

package SciSnet Types


All Types

Classes

Structures

Interfaces

Enumerat

Icon	Type	Description
	SN	SN Class - SciSnet Sockets Communication Interface Methods

Send comments on this topic to support@microglyph.com

(c) 2007 MicroGlyph Systems. All rights reserved.

SN Constructor

[Namespaces](#) > [SciSnet](#) > [SN](#) > [SN\(\)](#)

SN Class - constructor

Syntax

J#

public **SN** () Assembly: SCISNET (Module: SCISNET)

Send comments on this topic to support@microglyph.com

(c) 2007 MicroGlyph Systems. All rights reserved.

NETCLS Method (netid, code)

Namespaces > SciSnet > SN > NETCLS(Int32, Int32)

Routine to close a network connection

Syntax

J#

```
public static int NETCLS (  
    int netid,  
    int code  
)
```

Parameters

netid (**Int32**)

Network connection ID
1 thru 100

code (**Int32**)

Function code

- 1 - Purge input and output messages
- 2 - Purge input, wait for output messages

Return Value

Status is returned as an int value.

- 0 - Normal return
- 1 - Invalid arguments
- 2 - System exception
- 3 - SN Class not instantiated
- 4 - Connection not open
- 5 - Server not running

Assembly: SCISNET (Module: SCISNET)

Send comments on this topic to support@microglyph.com

(c) 2007 MicroGlyph Systems. All rights reserved.

NETDTM Method (month, day, year, hour, min, sec, msec)

Namespaces > SciSnet > SN > NETDTM(Int32, Int32, Int32, Int32, Int32, Int32, Int32)



Routine to get date and time.

Syntax

J#

```
public static int NETDTM (  
    /** @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 - SN Class not instantiated

Assembly: SCISNET (Module: SCISNET)

Send comments on this topic to support@microglyph.com

(c) 2007 MicroGlyph Systems. All rights reserved.

NETENC Method (code, bytbuf, intbuf, svalue)

Namespaces > [SciSnet](#) > [SN](#) > [NETENC\(Int32, Byte\[\], Int32\[\], String\)](#)

Routine to encode bytes, integers, or strings

Syntax

J#

```
public static int NETENC (  
    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 - SN Class not instantiated

Assembly: SCISNET (Module: SCISNET)

Send comments on this topic to support@microglyph.com

(c) 2007 MicroGlyph Systems. All rights reserved.

NETERR Method (Istexc, Istrtn, Isterr)

Namespaces > [SciSnet](#) > [SN](#) > NETERR(String, String, Int32)

Routine to get error messages and error code.

[-] Syntax

J#

```
public static int NETERR (  
    /** @ref */String Istexc,  
    /** @ref */String Istrtn,  
    /** @ref */int Isterr
```

[-] Parameters

Istexc (**String**)

Last SciSnet Exception

Istrtn (**String**)

Last SciSnet routine in error

Isterr (**Int32**)

Last SciSnet routine error code

[-] Return Value

Status is returned as an int value.

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

[-] Remarks

Method clears current error messages and code.

Assembly: SCISNET (Module: SCISNET)

Send comments on this topic to support@microglyph.com

(c) 2007 MicroGlyph Systems. All rights reserved.

NETFLS Method (netid, code)

Namespaces > SciSnet > SN > NETFLS(Int32, Int32)

Routine to flush network messages

Syntax

J#

```
public static int NETFLS (  
    int netid,  
    int code  
)
```

Parameters

netid (**Int32**)

Network connection ID
1 thru 100

code (**Int32**)

Function code

- 1 - Flush receive messages
- 2 - Flush send messages
- 3 - Flush receive and send messages

Return Value

Status is returned as an int value.

- 0 - Normal return
- 1 - Invalid arguments
- 2 - System exception
- 3 - SN Class not instantiated
- 4 - Connection not open
- 5 - Server not running

Assembly: SCISNET (Module: SCISNET)

Send comments on this topic to support@microglyph.com

(c) 2007 MicroGlyph Systems. All rights reserved.

NETOPN Method (netid, rname, rport, rtime)

Namespaces > [SciSnet](#) > [SN](#) > NETOPN(Int32, String, Int32, Int32)

Routine to open a network connections

Syntax

J#

```
public static int NETOPN (  
    int netid,  
    String rname,  
    int rport,  
    int rtime
```

Parameters

netid (**Int32**)

Network connection ID (1 - 100)

rname (**String**)

Remote host name or IP address

rport (**Int32**)

Remote host receive port(1-65535)

rtime (**Int32**)

Remote host connection timeout(ms)

Return Value

Status is returned as an int value.

- 0 - Normal return
- 1 - Invalid arguments
- 2 - System exception
- 3 - SN Class not instantiated
- 4 - Connection already open
- 5 - Server not running
- 6 - Duplicate remote host name
- 7 - Remote host not found
- 8 - Remote host connection failed
- 9 - Remote host connection timed out

Assembly: SCISNET (Module: SCISNET)

Send comments on this topic to support@microglyph.com

(c) 2007 MicroGlyph Systems. All rights reserved.

NETREC Method (netid, code, timeout, outbuf, limit, nbytes)

Namespaces > SciSnet > SN > NETREC(Int32, Int32, Int32, Byte[], Int32, Int32)



Routine to receive a message buffer

Syntax

J#

```
public static int NETREC (  
    int netid,  
    int code,  
    int timeout,  
    byte[] outbuf,  
    int limit,  
    /** @ref */int nbytes
```

Parameters

netid (**Int32**)

Network connection ID
1 thru 100

code (**Int32**)

Option code

- 1 - Normal read
- 2 - Read with timeout

timeout (**Int32**)

Timeout value (ms)

outbuf (**Byte[]**)

Output message buffer

limit (**Int32**)

Maximum length for outbuf

nbytes (**Int32**)

Number of bytes received

Return Value

Status is returned as an int value.

- 0 - Normal return
- 1 - Invalid arguments
- 2 - System exception
- 3 - SN Class not instantiated
- 4 - Connection not open
- 5 - Message truncated to limit
- 6 - Transmission errors detected
- 7 - Receive timed out
- 8 - Server not running

Assembly: SCISNET (Module: SCISNET)

Send comments on this topic to support@microglyph.com

(c) 2007 MicroGlyph Systems. All rights reserved.

NETSND Method (netid, code, outbuf, nbytes)

Namespaces > [SciSnet](#) > [SN](#) > [NETSND\(Int32, Int32, Byte\[\], Int32\)](#)

Routine to send a message buffer

Syntax

J#

```
public static int NETSND (  
    int netid,  
    int code,  
    byte[] outbuf,  
    int nbytes
```

Parameters

netid (**Int32**)

Network connection ID
1 thru 100

code (**Int32**)

Option code

- 1 - Send message
- 2 - Wait on message to be sent
- 3 - Timestamp message
- 4 - Wait + timestamp

outbuf (**Byte[]**)

Message buffer bytes array

nbytes (**Int32**)

Number of bytes to send

Return Value

Status is returned as an int value.

- 0 - Normal return
- 1 - Invalid arguments
- 2 - System exception
- 3 - SN Class not instantiated
- 4 - Connection not open
- 5 - Memory exhausted
- 6 - Transmission errors detected
- 7 - Server not running

Assembly: SCISNET (Module: SCISNET)

Send comments on this topic to support@microglyph.com

(c) 2007 MicroGlyph Systems. All rights reserved.

NETSRV Method (code, lport, rlen, slen)

Namespaces > SciSnet > SN > NETSRV(Int32, Int32, Int32, Int32)

Routine to start/stop local host server

Syntax

J#

```
public static int NETSRV (  
    int code,  
    int lport,  
    int rlen,  
    int slen  
)
```

Parameters

code (**Int32**)

Function code

- 1 - Start server
- 2 - Resume server
- 3 - Stop server

lport (**Int32**)

Local server port (1 - 65535)

rlen (**Int32**)

Receive TCP/IP buffer size

slen (**Int32**)

Send TCP/IP buffer size

Return Value

Status is returned as an int value.

- 0 - Normal return
- 1 - Invalid arguments
- 2 - System exception
- 3 - SN Class not instantiated
- 4 - Server already started
- 5 - Server already stopped
- 6 - Receive thread failed to start
- 7 - Send thread failed to start
- 8 - Timeout thread failed to start
- 9 - Receive thread failed to stop
- 0 - Send thread failed to stop
- 1 - Timeout thread failed to stop
- 2 - Memory allocation error

Assembly: SCISNET (Module: SCISNET)

Send comments on this topic to support@microglyph.com

(c) 2007 MicroGlyph Systems. All rights reserved.

NETSTS Method (netid, lname, laddr, rname, raddr, nrmsg, nsmsg, nrcnt, nscnt, nrerr, nserr, rerr, serr)

Namespaces > SciSnet > SN > NETSTS(Int32, String, String, String, String, Int32, Int32,

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

Routine to get network connection status

Syntax

J#

```
public static int NETSTS (  
    int netid,  
    /** @ref */String lname,  
    /** @ref */String laddr,  
    /** @ref */String rname,  
    /** @ref */String raddr,  
    /** @ref */int nrmsg,  
    /** @ref */int nsmsg,  
    /** @ref */int nrcnt,  
    /** @ref */int nscnt,  
    /** @ref */int nrerr,  
    /** @ref */int nserr,  
    /** @ref */int rerr,  
    /** @ref */int serr  
)
```

Parameters

netid (**Int32**)

Network connection ID (1-100)

lname (**String**)

Local host name

laddr (**String**)

Local host ip address

rname (**String**)

Remote host name

raddr (**String**)

Remote host ip address

nrmsg (**Int32**)

Number of receive messages

nsmsg (**Int32**)

Number of send messages

nrcnt (**Int32**)

Number bytes receive msgs

nscnt (**Int32**)

Number bytes send msgs

nrerr (**Int32**)

Number receive msgs errors

nserr (**Int32**)

Number send msgs errors

rerr (**Int32**)

Receive message error

serr (**Int32**)

Send message error

Return Value

Status is returned as an int value.

- 0 - Normal return
- 1 - Invalid arguments

- 2 - System exception
- 3 - SN Class not instantiated
- 4 - Connection not open
- 5 - Transmission errors detected
- 6 - Server not running

Assembly: SCISNET (Module: SCISNET)

Send comments on this topic to support@microglyph.com

(c) 2007 MicroGlyph Systems. All rights reserved.

NETSUS Method (tmeout)

Namespaces > SciSnet > SN > NETSUS(Int32)

Routine to sleep the executing thread.

Syntax

J#

```
public static int NETSUS (  
    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 - SN Class not instantiated

Assembly: SCISNET (Module: SCISNET)

Send comments on this topic to support@microglyph.com

(c) 2007 MicroGlyph Systems. All rights reserved.

NETTMR Method

[Namespaces](#) > [SciSnet](#) > [SN](#) > [NETTMR\(\)](#)

Routine to get current time in milliseconds.

Syntax

J#

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

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

Assembly: SCISNET (Module: SCISNET)

Send comments on this topic to support@microglyph.com

(c) 2007 MicroGlyph Systems. All rights reserved.

Example Programs

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

The following links provide access to the example programs:

[TNET](#)

Use of SciComm Library Methods.

```

/*****
 *
 *   TNET - Program to test .NET 2005 TCP/IP Sockets support
 *
 *****/

import System.*;
import System.Threading.*;
import SciSnet.*;

class TNET
{
/* Instantiate SciSnet Class */

    public static SN SNClass = new SN();

/*   Main Program */

/**  @attribute STAThread() */

    public static void main(String[] args)
    {

/*   Local variables */

        int status,i = 0,j = 0,NetID = 0,LPort = 0,RPort = 0,RTime = 0,
            Rlen = 0,Slen = 0,Code = 0,Nrmsg = 0,Nsmsg = 0,Nrcnt = 0,
            Nscnt = 0,Nrerr = 0,Nserr = 0,Rerr = 0,Serr = 0,Lsterr = 0,
            ival = 0,NBytes = 0,Func = 0,Limit = 80,Value = 0,j1 = 0,
            j2 = 0,n = 0;
        int[] IntBuf = new int[16];
        ubyte[] TmpBuf;
        ubyte[] BytBuf;
        char reply;
        String RName,Rname = "",Raddr = "",Lname = "",Laddr = "",
            Lstexc = "",Lstrtn = "",line = "", pline = "";
        boolean dialog, SFlag;

/*   Start Main */

        try
        {

/*   Dialog processing loop */

            dialog = true;
            SFlag = false;
            while (dialog)
            {

/*   Main Function Message */

                Console.WriteLine("\n TNET Functions:\n");
                Console.WriteLine("   N = Server      (NETSRV)");
                Console.WriteLine("   O = Open       (NETOPN)");
                Console.WriteLine("   C = Close      (NETCLS)");
                Console.WriteLine("   S = Status     (NETSTS)");
                Console.WriteLine("   R = Read       (NETREC)");
            }
        }
    }
}

```

```

Console.WriteLine("    W = Write          (NETSND)");
Console.WriteLine("    F = Flush          (NETFLS)");
Console.WriteLine("    M = Display Menu");
Console.WriteLine("    Q = Quit/Exit");
Console.Write("\n Enter Function: ");
/*
Get function code
line = Console.ReadLine().ToUpper();
if (line.get_Length() == 0)
{
/*
    Nothing entered, loop
    continue;
}
reply = line.get_Chars(0);

/*
Process reply

switch (reply)
{

case ' ':
/*
    Blank line
    break;

case 'M':
/*
    Menu command

    break;

case 'N':
/*
    Network server

    Console.WriteLine("\n    Network Server Functions:\n");
    Console.WriteLine("        1 = Start Network Server");
    Console.WriteLine("        2 = Resume Network Server");
    Console.WriteLine("        3 = Stop Network Server");
    if (SFlag)
    {
/*
        Network server is started
        Code = 3;
        Console.Write("\n            Enter Function [3]: ");
        SFlag = false;
    }
    else
    {
/*
        Network server is down
        Code = 1;
        Console.Write(
            "\n            Enter Function
            [1]: ");
        SFlag = true;
    }
    line = Console.ReadLine();
    if (line.get_Length() > 0)
    {
/*
        Int32.TryParse(line,ival);
        Set select code value
        Code = ival;
    }
}

```

```

/*      Start Network Server parameters      */
if (Code == 1)
{
/*      Set local listen port      */
LPort = 8001;
Console.Write(
    "      Enter Local Listen Port      [8001]: ");
line = Console.ReadLine();
if (line.get_Length() > 0)
{
/*      Set selected local port value      */
LPort = ival;
}
/*      Set TCP/IP receive buffer length      */
Rlen = 1440;
Console.Write(
    "      Enter Receive Buffer Length[1440]: ");
line = Console.ReadLine();
if (line.get_Length() > 0)
{
/*      Set selected receive buffer length      */
Rlen = ival;
}
/*      Set TCP/IP send buffer length      */
Slen = 1440;
Console.Write(
    "      Enter Send Buffer Length      [1440]: ");
line = Console.ReadLine();
if (line.get_Length() > 0)
{
/*      Set selected send buffer length      */
Slen = ival;
}
}
/*      Call NETSRV routine      */
status = SN.NETSRV(Code,LPort,Rlen,Slen);
if (status != 0)
{
    Console.WriteLine("\n NETSRV error: ({0})\n",
        (Int32)status);
    Console.WriteLine("      Code      = {0}",(Int32)Code);
    Console.WriteLine("      LPort     = {0}",(Int32)LPort);
    Console.WriteLine("      Rlen      = {0}",(Int32)Rlen);
    Console.WriteLine("      Slen      = {0}",(Int32)Slen);
}
break;

case 'O':

/*      Open network connection      */

/*      Get NetID      */
NetID = 1;
Console.Write(
    "\n      Enter Network Connection ID      [1]: ");
line = Console.ReadLine();
if (line.get_Length() > 0)

```

```

        {
            Int32.TryParse(line,ival);
/*          Set NetID value                                     */
            NetID = ival;
        }
/*      Get remote host name                                   */
        Console.Write(
            "          Enter Remote Host Name                : ");
        line = Console.ReadLine();
        if (line.get_Length() > 0)
        {
/*          Set remote host name                               */
            RName = line;
        }
        else
        {
/*          Nothing entered                                   */
            continue;
        }
/*      Get remote port                                       */
        RPort = 8001;
        Console.Write(
            "          Enter Remote Host Receive Port        [8001]: ");
        line = Console.ReadLine();
        if (line.get_Length() > 0)
        {
/*          Int32.TryParse(line,ival);                         */
            Int32.TryParse(line,ival);
/*          Set selected remote port value                    */
            RPort = ival;
        }
/*      Get remote host connect timeout value                 */
        RTime = 0;
        Console.Write(
            "          Enter Remote Host Connect Timeout Value[0]: ");
        line = Console.ReadLine();
        if (line.get_Length() > 0)
        {
/*          Int32.TryParse(line,ival);                         */
            Int32.TryParse(line,ival);
/*          Set remote host connect timeout value            */
            RTime = ival;
        }
/*      Open connection to remote host                         */
        status = SN.NETOPN(NetID,RName,RPort,RTime);
        if (status != 0)
        {
            Console.WriteLine("\n NETOPN error: ({0})\n",
                (Int32)status);
            Console.WriteLine("          NetID = {0}",(Int32)NetID);
            Console.WriteLine("          RName = {0}",RName);
            Console.WriteLine("          RPort = {0}",(Int32)RPort);
            Console.WriteLine("          RTime = {0}",(Int32)RTime);
        }
        break;

    case 'C':

/*      Close network connection                               */

/*      Get function code                                     */
        Console.WriteLine("\n          Close Functions:\n");

```

```

Console.WriteLine(
    "          1 = Purge input and output messages");
Console.WriteLine(
    "          2 = Purge input,wait for output messages");
Console.Write("\n          Enter Function          [1]: ");
Code = 1;
line = Console.ReadLine();
if (line.get_Length() > 0)
    {
        Int32.TryParse(line,ival);
/*          Set select code value          */
        Code = ival;
    }
/*          Get NetID          */
NetID = 1;
Console.Write("          Enter Network Connection ID[1]: ");
line = Console.ReadLine();
if (line.get_Length() > 0)
    {
        Int32.TryParse(line,ival);
/*          Set NetID value          */
        NetID = ival;
    }
/*          Close network connection          */
status = SN.NETCLS(NetID,Code);
if (status != 0)
    {
        Console.WriteLine("\n NETCLS error: ({0})\n",
            (Int32)status);
        Console.WriteLine("          NetID = {0}",(Int32)NetID);
        Console.WriteLine("          Code = {0}",(Int32)Code);
    }
break;

case 'S':

/*          Network connection status          */

/*          Get NetID          */
NetID = 1;
Console.Write("\n          Enter Network Connection ID[1]: ");
line = Console.ReadLine();
if (line.get_Length() > 0)
    {
        Int32.TryParse(line,ival);
/*          Set NetID value          */
        NetID = ival;
    }
/*          Get status for this network connection          */
status = SN.NETSTS(NetID,Lname,Laddr,Rname,Raddr,Nrmsg,
    Nsmmsg,Nrcnt,Nscnt,Nrerr,Nserr,Rerr,Serr);

if (status == 0)
    {
        Console.WriteLine("\n          NetID = {0}",
            (Int32)NetID);
        Console.WriteLine("          Lname = {0}",Lname);
        Console.WriteLine("          Laddr = {0}",Laddr);
        Console.WriteLine("          Rname = {0}",Rname);
        Console.WriteLine("          Raddr = {0}",Raddr);
    }

```

```

        Console.WriteLine("          Nrmsg = {0}", (Int32)Nrmsg);
        Console.WriteLine("          Nsmg  = {0}", (Int32)Nsmg);
        Console.WriteLine("          Nrcnt = {0}", (Int32)Nrcnt);
        Console.WriteLine("          Nscnt = {0}", (Int32)Nscnt);
        Console.WriteLine("          Nrerr = {0}", (Int32)Nrerr);
        Console.WriteLine("          Nserr = {0}", (Int32)Nserr);
        Console.WriteLine("          Rerr  = {0}", (Int32)Rerr);
        Console.WriteLine("          Serr  = {0}", (Int32)Serr);
    }
else
    {
        Console.WriteLine("\n NETSTS error: ({0})\n",
            (Int32)status);
        Console.WriteLine("          NetID = {0}", (Int32)NetID);
        Console.WriteLine("          Lname = {0}", Lname);
        Console.WriteLine("          Laddr = {0}", Laddr);
        Console.WriteLine("          Rname = {0}", Rname);
        Console.WriteLine("          Raddr = {0}", Raddr);
        Console.WriteLine("          Nrmsg = {0}", (Int32)Nrmsg);
        Console.WriteLine("          Nsmg  = {0}", (Int32)Nsmg);
        Console.WriteLine("          Nrcnt = {0}", (Int32)Nrcnt);
        Console.WriteLine("          Nscnt = {0}", (Int32)Nscnt);
        Console.WriteLine("          Nrerr = {0}", (Int32)Nrerr);
        Console.WriteLine("          Nserr = {0}", (Int32)Nserr);
        Console.WriteLine("          Rerr  = {0}", (Int32)Rerr);
        Console.WriteLine("          Serr  = {0}", (Int32)Serr);
    }
break;

case 'R':

/*      Read bytes from receive buffer      */

/*      Get function code                    */
Console.WriteLine("\n      Receive Functions:\n");
Console.WriteLine("          1 = Normal Read");
Console.WriteLine("          2 = Read with Timeout");
Console.Write("\n      Enter Function          [1]: ");
Code = 1;
line = Console.ReadLine();
if (line.get_Length() > 0)
    {
        Int32.TryParse(line, ival);
/*      Set select code value                */
Code = ival;
    }

/*      Get timeout value                    */
Value = 0;
if (Code == 2)
    {
        Console.Write("\n      Enter Read Timeout Value: ");
line = Console.ReadLine();
if (line.get_Length() > 0)
        {
            Int32.TryParse(line, ival);
/*      Set read timeout value                */
Value = ival;
        }
    }

/*      Get NetID                            */

```

```

NetID = 1;
Console.WriteLine("      Enter Network Connection ID[1]: ");
line = Console.ReadLine();
if (line.get_Length() > 0)
    {
        Int32.TryParse(line,ival);
/*      Set NetID value      */
        NetID = ival;
    }
/*  Read a message from a remote host  */
TmpBuf = new ubyte[Limit];
status = SN.NETREC(NetID,Code,Value,TmpBuf,Limit,NBytes);
if (status == 0)
    {
        if (NBytes == 0)
            {
                Console.WriteLine("\n NETREC: No Data Available\n");
                Console.WriteLine("      NetID  = {0}",(Int32)NetID);
                Console.WriteLine("      Code   = {0}",(Int32)Code);
            }
        if (NBytes > 0)
            {
                BytBuf = new ubyte[NBytes];
/*      Move number of bytes read into byte buffer      */
                for (i = 0; i < NBytes; i++)
                    {
                        BytBuf[i] = TmpBuf[i];
                    }
                Console.WriteLine("\n      Count: {0}",(Int32)NBytes);
/*      Convert bytes received to ASCII string      */
                Func    = 5;
                line    = "";
                status = SN.NETENC(Func,BytBuf,IntBuf,line);
/*      Display ASCII string      */
                for (i = 0; i < NBytes; i=i+32)
                    {
                        n = System.Math.Min(32,NBytes-i);
                        pline = line.Substring(i,n);
                        if (i == 0)
                            {
                                Console.WriteLine(
                                    "\n      ASCII: {0}",pline);
                            }
                        else
                            {
                                Console.WriteLine(
                                    "      : {0}",pline);
                            }
                    }
                }
/*      Display Hex string      */
                for (i = 0; i < NBytes; i=i+11)
                    {
                        j1 = i;
                        j2 = i + System.Math.Min(11,NBytes-i);
                        if (i == 0)
                            {
                                Console.WriteLine("      Hex  : ");
                                for (j = j1; j < j2; j++)
                                    {
                                        Console.WriteLine("{0,2:X2} ",(Int32)BytBuf[j]);
                                    }
                            }
                    }
            }
    }

```

```

        }
        Console.WriteLine(" ");
    }
    else
    {
        Console.Write("                : ");
        for (j = j1; j < j2; j++)
        {
            Console.Write("{0,2:X2} ", (Int32)BytBuf[j]);
        }
        Console.WriteLine(" ");
    }
}
}
else
{
    Console.WriteLine("\n NETREC error: ({0})\n",
        (Int32)status);
    Console.WriteLine("    NetID = {0}", (Int32)NetID);
    Console.WriteLine("    Code = {0}", (Int32)Code);
    Console.WriteLine("    Value = {0}", (Int32)Value);
}
break;

case 'W':

/*      Write out bytes to send buffer      */

/*      Get function code      */
Console.WriteLine("\n    Send Functions:\n");
Console.WriteLine("    1 = Send Message");
Console.WriteLine("    2 = Wait on Message to be sent");
Console.WriteLine("    3 = Timestamp Message");
Console.WriteLine("    4 = Wait + Timestamp");
Console.Write("\n    Enter Function          [1]: ");
Code = 1;
line = Console.ReadLine();
if (line.get_Length() > 0)
{
/*      Set select code value      */
    Code = ival;
}

/*      Get NetID      */
NetID = 1;
Console.Write("    Enter Network Connection ID[1]: ");
line = Console.ReadLine();
if (line.get_Length() > 0)
{
/*      Set NetID value      */
    NetID = ival;
}

/*      Get output line      */
Console.Write("    Enter Output String          : ");
line = Console.ReadLine();
if (line.get_Length() == 0)
{
/*      Nothing to send, loop      */

```

```

        continue;
    }
    /* Convert string to ASCII bytes */
    Func    = 3;
    NBytes  = line.get_Length();
    BytBuf  = new ubyte[NBytes];
    status  = SN.NETENC(Func,BytBuf,IntBuf,line);
    /* Send ASCII bytes */
    status  = SN.NETSND(NetID,Code,BytBuf,NBytes);
    if (status != 0)
    {
        Console.WriteLine("\n NETSND error: ({0})\n",
            (Int32)status);
        Console.WriteLine("    NetID    = {0}",(Int32)NetID);
        Console.WriteLine("    Code     = {0}",(Int32)Code);
        Console.WriteLine("    line     = {0}",line);
        Console.WriteLine("    NBytes   = {0}",(Int32)NBytes);
    }
    break;

case 'F':

    /* Flush receive and send buffers */

    /* Get function code */
    Console.WriteLine("\n    Flush Functions:\n");
    Console.WriteLine("    1 = Flush Receive Messages");
    Console.WriteLine("    2 = Flush Send Messages");
    Console.WriteLine("    3 = Flush Receive + Send Messages");
    Console.Write("\n    Enter Function          [1]: ");
    Code = 1;
    line = Console.ReadLine();
    if (line.get_Length() > 0)
    {
        Int32.TryParse(line,ival);
        /* Set select code value */
        Code = ival;
    }

    /* Get NetID */
    NetID = 1;
    Console.Write("    Enter Network Connection ID[1]: ");
    line = Console.ReadLine();
    if (line.get_Length() > 0)
    {
        Int32.TryParse(line,ival);
        /* Set NetID value */
        NetID = ival;
    }

    /* Flush buffers */
    status = SN.NETFLS(NetID,Code);
    if (status != 0)
    {
        Console.WriteLine("\n NETFLS error: ({0})\n",
            (Int32)status);
        Console.WriteLine("    NetID    = {0}",(Int32)NetID);
        Console.WriteLine("    Code     = {0}",(Int32)Code);
    }
    break;

case 'Q':

```

```

/*      Exit TNET program                                     */
        dialog = false;
        break;

        default:

/*      Unknown command                                     */

        break;

    }

/*      Dialog process loop                                 */

    }

/*      Report any errors                                   */

    status = SN.NETERR(Lstexc,Lstrtn,Lsterr);
    if (Lsterr == 0)
    {
/*      Report normal completion                             */
        Console.WriteLine("\n  Normal completion\n");
    }
    else
    {
/*      Report Last Errors                                   */
        Console.WriteLine("\n Last Exception           : {0}",Lstexc);
        Console.WriteLine(" Last Member in Error      : {0}",Lstrtn);
        Console.WriteLine(" Last Member Error Code : {0}",
            (Int32)Lsterr);
    }

    }

    catch (System.FormatException e)
    {
/*      Report exception in Main Program                     */
        Console.WriteLine(" Exception {0} ",e);
        return;
    }

    catch (System.Exception e)
    {
/*      Report exception in Main Program                     */
        Console.WriteLine(" Exception {0} ",e);
        return;
    }

/*      Exit application                                     */

    return;

}
}

```