Network Serial Port Applications Programming Interface Programmers Guide and Reference

Constellation Data Systems, Inc.
www.VirtualPeripherals.com

Table of Contents

1. Introduction

The *NSP Pro* Applications Programming Interface (API) allows rapid software development, and is therefore a development accelerator, which can cut months or years from a development project, which requires a local or remote Virtual Serial Port (VSP) or Physical Serial Port (PSP) resource.

1.1 What's in This Manual

This manual describes the Applications Programming Interface (API) of the Virtual Serial Port framework.

1.2 Audience

This literature is for use by the programmer who wishes to develop software, which interfaces with a VSP or PSP, which is expressed or accessed across a computer network, such as TCP/IP. It is assumed that the reader is a skilled programmer, with a basic understanding of Windows programming, and serial communications in the Windows environment.

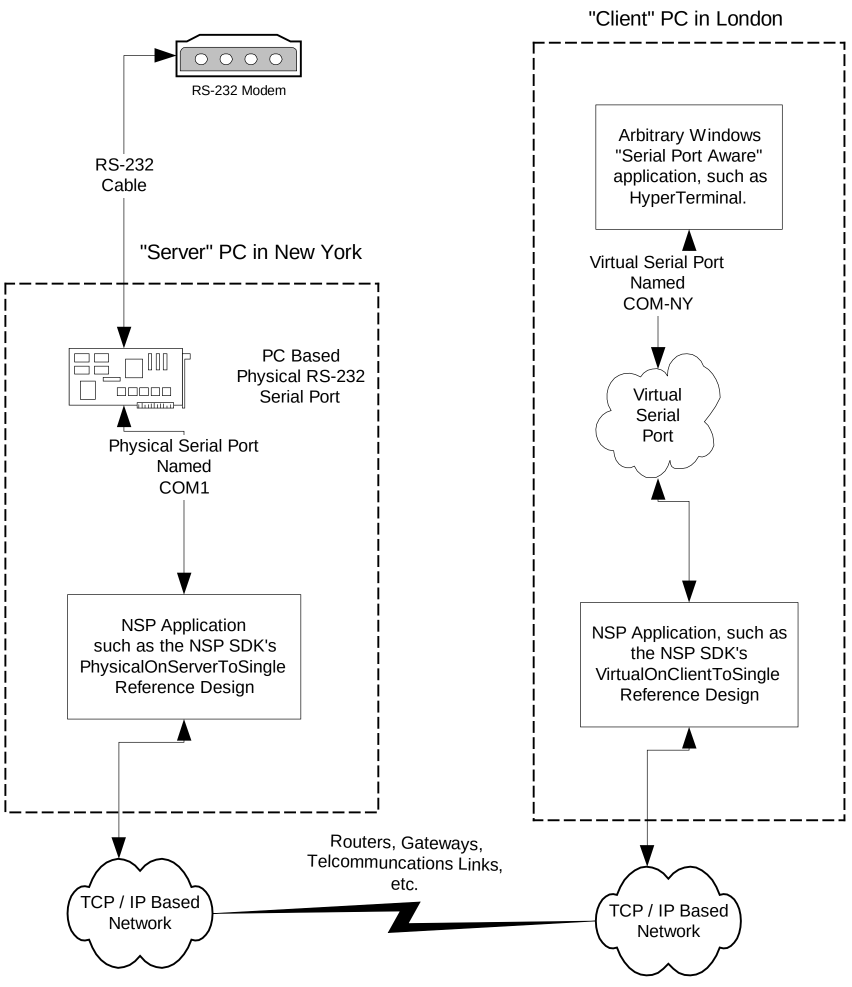
Applications, which support the NSP API, are simply called "NSP Applications". Constellation Data Systems, Inc., (CDS) distributes several NSP Application reference design applications as components of the NSP Software Development Kit (SDK). These reference designs are fully functional, and are distributed in source format with the SDK. You may wish to use one of these frameworks as a starting point for your development.

1.3 Limitations

Use of this software, information, or technology in a system, or as a component of a system, which can through action or inaction, cause damage to life, limb, property, or the environment is not authorized. Use of this software is also subject to the terms and conditions of your properly executed Software License Agreement(s) with CDS.

2. Hypothetical NSP Based Implementation

In the following hypothetical NSP example, a serial device (in this case a modem) has been expressed across a network.


"Client" PC in London
RS-232 Modem
RS-232 Cable
Arbitrary Windows "Serial Port Aware" application, such as HyperTerminal.
"Server" PC in New York
Virtual Serial Port Named COM-NY
PC Based Physical RS-232 Serial Port
Virtual Serial Port
Physical Serial Port Named COM1
NSP Application such as the NSP SDK's PhysicalOnServerToSingle Reference Design
NSP Application, such as the NSP SDK's VirtualOnClientToSingle Reference Design
Routers, Gateways, Telcommuncations Links, etc.
TCP / IP Based Network
TCP / IP Based Network

The user of the PC in London can access a Serial Port, named (for example), COM-PARIS, and access that device as though it were a local device.

3. Binding the NSP API to Your Application

3.1 Introduction – C++ Class form

The VSP API is implemented using a C++ class interface which conforms to name decoration rules used by the Microsoft Visual C / C++ compiler, version 6.0.

3.2 Source Code Include (“NspApi.h”)

The NSP API class interface is stored in “NspApi.h”. Applications typically use the following include identifies the interface:

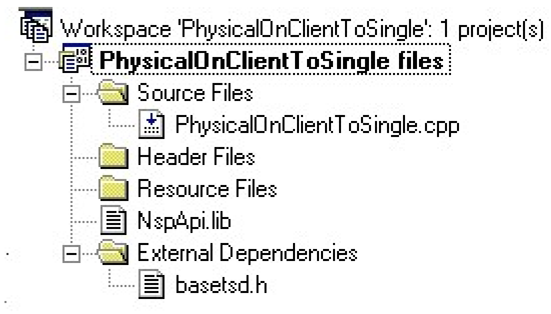
#include "..\NspApi\NspApi.h"

3.3 The “NspApi.dll” File

At run time the “NspApi.dll” must be found at the time an NSP Application is started. It is recommended that “NspApi.dll” reside in the same directory as the NSP Application’s executable.

3.4 Binding “NspApi.lib” File

In order for the DLL to bind to the application, the “NspApi.lib” file must be included in the link sequence. For that reason, NSP Applications include “NspApi.lib” in the applications resources. Consider the following screen capture from the Microsoft Visual C/C++ Developer Studio:


Workspace 'PhysicalOnClientToSingle': 1 project(s)
PhysicalOnClientToSingle files
Source Files
PhysicalOnClientToSingle.cpp
Header Files
Resource Files
NspApi.lib
External Dependencies
basetsd.h

4. NSP API Reference (C++ Class Form)

4.1 Overview of the NSP API

Underlying the NSP API is the industry standard TCP/IP Sockets interface for network communication. A basic understanding of the Sockets terminology of "*Client*" and "*Server*" access methods is necessary to use the API.

Important Point #1	In the jargon of Sockets, the system that answers a connection (like answering an incoming telephone call), is called the *"Server".* Analogously, the system that initiates a connection (like dialing a telephone call) is called the *"Client".*
Important Point #2	The *Client* application connects to the remote server using the *Server's Network Address* (generally an IP address specified in dotted quad format) and a *Socket Port* number (an integer). Using the telephone analogy, the *Client* "dials" the telephone number (IP address / port number) of the server.
Important Point #3	The *Server* application accepts connections from the remote *Client* using a *Socket Port* number only. The IP address is not needed by the *Server,* because the server connections are always accepted by the *Server* at its IP address. Using the telephone analogy, the *Server* "answers" the telephone call initiated by the *Client.*

The cNspApi Class encapsulates lower level driver and system interface techniques into an environment, which is both simple and powerful.

4.2 Constructor / Destructors of cNspApi

The constructor and destructor have the following forms:

```
cNspApi(void);
~cNspApi(void);
```

Constructing an NSP class instance prepares the underlying data structures.

4.3 OpenConnectionToClient ( ) Function of cNspApi

The *OpenConnectionToClient* function is intended for execution on an NSP Application running on a machine acting as a *Server*. This function will block until a *Client* attempts connection, then accepts a single connection (from the remote *Client*), on the *Socket Port* specified.

When connections created by *OpenConnectionToClient* are no longer desired, the NSP Application should execute the *CloseConnection* NSP API function on the target connection.

Prototype

int OpenConnectionToClient (int iSocketPort,
*int *pConnection);*

Parameters

IsocketPort	*Socket Port* number (integer) on the local host (server) machine.
PConnection	When connected, the integer addressed by *pConnection* identifies the connection. The caller should save this value because it identifies the connection and is used by other NSP API functions.

Return Values

If the function succeeds, the return value is zero. Other status returns are defined by the MS Platform SDK files: “winerror.h” and “winsock2.h” .

Also See

CloseConnection ( )

4.4 OpenConnectionToServer ( ) Function of cNspApi

The *OpenConnectionToServer* function is intended for execution on an NSP Application running on a machine acting as a *Client*. This function connects the local NSP application ("*Client")* to the remote *Server* on the Network Address and *Socket Port* specified.

When connections created by *OpenConnectionToServer* are no longer desired, the NSP Application should execute the *CloseConnection* NSP API function on the target connection.

Prototype

```
int OpenConnectionToServer  (char *szNetworkAddress,
                             int iSocketPort, int *pConnection);
```

Parameters

szNetworkAddress	Remote server *Network Address* (generally an IP address in "dotted quad" format) in plain text, which is a target of the connect operation.
IsocketPort	*Socket Port* number of the remote server machine which is a target of the connect operation.
pConnection	When connected, the integer addressed by *pConnection* identifies the connection. The caller should save this value because it identifies the connection and is used by other NSP API functions.

Return Values

If the function succeeds, the return value is zero. Other status returns are defined by the MS Platform SDK files: "winerror.h" and "winsock2.h" .

Also See

CloseConnection ( )

4.5 CloseConnection ( ) Function of cNspApi

The *CloseConnection* function releases a previously opened connection NSP API connection. This function is intended for execution on an NSP Application running on machines acting either *Server* or *Client* mode.

Each connection established by NSP API “open” function (those functions which are named prefaced by “*OpenConnection”) is* closed (when the connection is no longer desired), by using the *CloseConnection* function.

Prototype

int CloseConnection (int Connection);

Parameters

Connection	An integer which identifies the connection to be closed.

Return Values

If the function succeeds, the return value is zero. Other status returns are defined by the MS Platform SDK files: “winerror.h” and “winsock2.h” .

Also See

OpenConnectionToServer ( )
OpenConnectionToClient ( )

4.6 Read ( ) Function of cNspApi

The *Read* function reads data from the target connection. This function blocks until data is available to be read from the connection.

The number of bytes actually read by this connection depends upon the number of bytes received by the connection. It is therefore expected that the number of bytes read to generally be less than the number of bytes requested (in the *SizeofBuff* parameter).

Prototype

```
int Read     (int Connection, char *pBuff,
              int SizeofBuff, int *pBytesRead);
```

Parameters

Connection	An integer which identifies the connection from which the data will be read.
pBuff	Pointer to buffer which receives the data read from the connection
SizeofBuff	Maximum amount of data which can be read to "pBuff".
pBytesRead	Pointer to an integer, which contains the number of bytes actually read from the connection.

Return Values

If the function succeeds, the return value is zero. Other status returns are defined by the MS Platform SDK files: "winerror.h" and "winsock2.h" .

Also See

OpenConnectionToServer ( )
OpenConnectionToClient ( )

4.7 Write ( ) Function of cNspApi

The *Write* function writes data to the target connection. This function does not block, and is expected to return immediately.

Prototype

*in*t Write *(int Connection, char *pBuff, int SizeofToWrite);*

Parameters

Connection	An integer which identifies the connection which will receive the data written.
pBuff	Pointer to buffer which contains the data to be written to the connection
SizeofToWrite	Number of bytes to write from *pBuff* to the *Connection*.

Return Values

If the function succeeds, the return value is zero. Other status returns are defined by the MS Platform SDK files: “winerror.h” and “winsock2.h” .

Also See

Read ( )

4.8 DllVersion ( ) Function of cNspApi

The *DllVersion* function returns the version number of the underlying "NspApi.dll" file. The version number returned is the actual version number, multiplied by 100. For example, version 2.08 of NspApi.dll would be returned as a value of 208.

Prototype

```
int DllVersion (ULONG *pVersion);
```

Parameters

pVersion	An unsigned long integer, which receives the version number of the underlying NspApi.dll file.

Return Values

If the function succeeds, the return value is zero. Other status returns are defined by the MS Platform SDK files: "winerror.h" and "winsock2.h" .

4.9 QueryConnectionInfo ( ) Function of cNspApi

The *QueryConnectionInfo* function returns the *Network Address* (generally the IP address) and port number of the connection. This function is expected to be used by *Server* NSP applications to identify the connected *Client*.

Prototype

```
int QueryConnectionInfo (int Connection,
                         unsigned int *IpAddress,
                         unsigned short *Port);
```

Parameters

Connection	An integer which identifies the connection.
lpAddress	Pointer to a buffer the *Network Address*, in a unsigned integer (32) bits, which represents the target connection's IP address.
Port	*Socket Port* of the target connection.

Return Values

If the function succeeds, the return value is zero. Other status returns are defined by the MS Platform SDK files: "winerror.h" and "winsock2.h" .

Also See

OpenConnectionToServer ( )
OpenConnectionToClient ( )

4.10 SetDebugMode ( ) Function of cNspApi

The *SetDebugMode* function allows the developer to enable or disable specific debugging features of the framework. The *DebugMode* is initially set fully off (zero) until enabled.

Prototype

int SetDebugMode (int DebugMode);

Parameters

DebugMode	An integer with the following bit fields: DEBUG_MODE_MESSAGE_BOX – Enables message box debugging when specified. DEBUG_MODE_CONSOLE – Enables printf style console debugging.

Return Values

If the function succeeds, the return value is zero. Other status returns are defined by the MS Platform SDK files: “winerror.h” and “winsock2.h” .

4.11 IsConnectionValid ( ) Function of cNspApi

The *IsConnectionValid* function returns the status of the specified connection. This function is expected to be used by any NSP application to determine if a client or server has disconnected.

Prototype

```
int IsConnectionValid    (int Connection,
                         bool *bStatus);
```

Parameters

Connection	An integer which identifies the connection.
bStatus	Pointer to a boolean. TRUE – connection is still valid. FALSE – connection has been terminated.

Return Values

If the function succeeds, the return value is zero. Other status returns are defined by the MS Platform SDK files: “winerror.h” and “winsock2.h” .

Also See

OpenConnectionToServer ( )
OpenConnectionToClient ( )

5. Index of Acronyms and Abbreviations

API	Applications Programming Interface
cNspApi	Network Serial Port API Class
CDS	Constellation Data Systems
DLL	Dynamic Link Library
MS	Microsoft
MSDN	MS Developers Network
NSP	Network Serial Port
PC	Personal Computer
PCR	Physical Communications Resource (Such as a UART)
SDK	Systems Development Kit
TLA	Three Letter Acronym
VSP	Virtual Serial Port