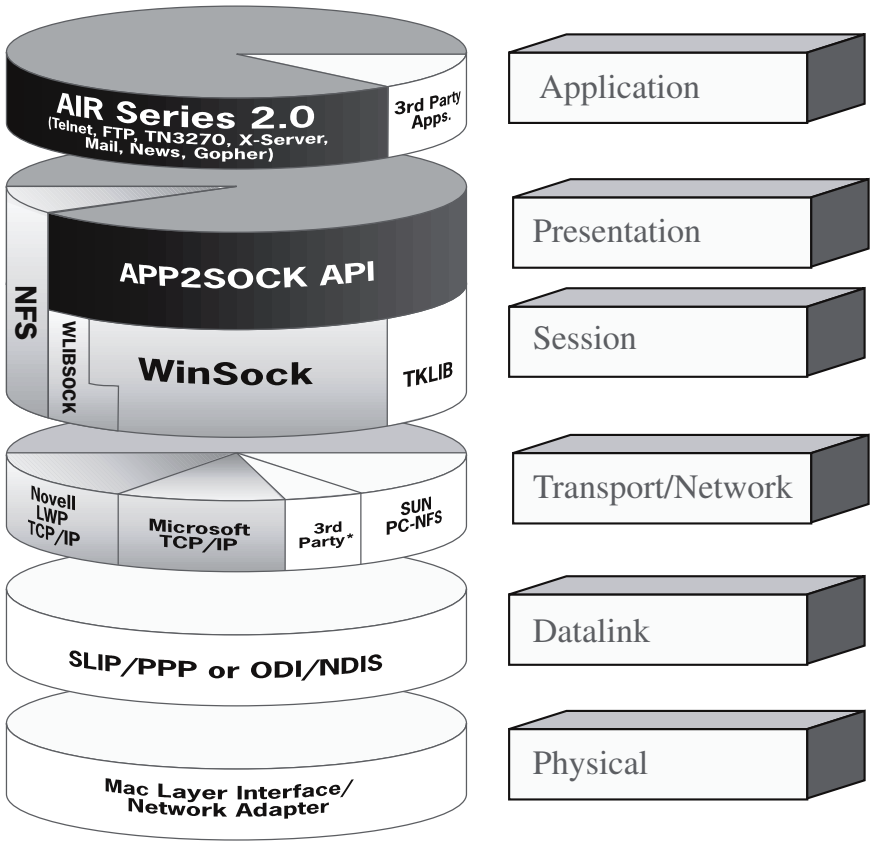
APP2SOCK

Network Independent Architecture



Technical Guide

Copyright

APP2SOCK and APP2SOCK Technical Guide (c) SPRY, Inc., 1993.

Questions?

Please direct any questions about the APP2SOCK API to **techs@spry.com**, or fax questions to the fax number provided below.

SPRY, Inc.
316 Occidental Ave S
Seattle, WA 98104
(206) 447-0300
(206) 447-9008 FAX
info@spry.com

Contents

Introduction 4
The APP2SOCK Developer's Kit 6
APP2SOCK Functions 7
COMMInitial() 8
COMMListen() 9
COMMOpen() 10
COMMTestConnected() 12
COMMAllowNewServer() 13
COMMAllowEvent() 14
COMMCharsPresent() 15
COMMRead() 16
COMMPeek() 17
COMMWriteCharacter() 18
COMMWriteBlock() 19
COMMWriteASCIIString() 20
COMMClose() 21
COMMGetPeerInfo() 22
COMMGetMyIP() 23
COMMGetErrorMessageString() 24
COMMFakeOpen() 25
COMMFakeClose() 26
COMMWaitForData() 27
Sample APP2SOCK Code 28

Introduction

APP2SOCK (*Application to Socket Interface*) is a public specification providing a high-layer API which bridges Windows TCP/IP applications to many of the most common TCP/IP transports. APP2SOCK supports the evolving WinSock standard as well as Novell's WLIBSOCK and SunSelect's TKLIB Dynamic Link Libraries (DLL). APP2SOCK also has the ability to support other proprietary socket libraries.

APP2SOCK was developed for use with SPRY's **AIR Series** of TCP/IP applications for Windows, making the AIR applications portable to multiple socket interfaces, as shown below.

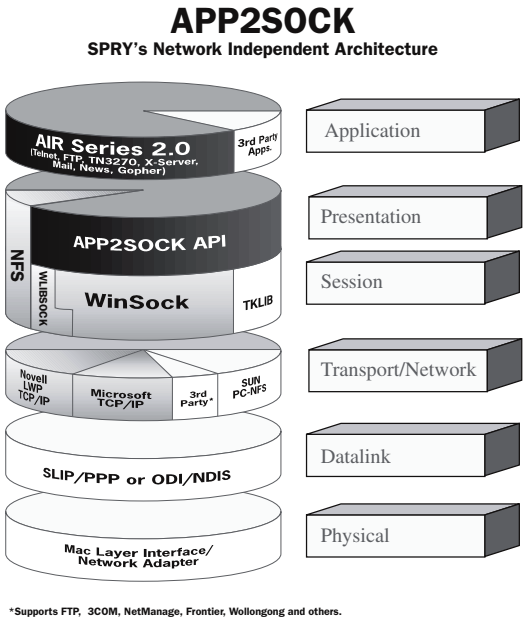


Diagram of the APP2SOCK Architecture in the seven layer OSI model

At SPRY we believe that your applications should define your network, not the protocols and other drivers beneath them. Our network independent architecture, APP2SOCK, enables you to focus on implementing our easy-to-use applications rather than wrestling with underlying compatibility issues.

TCP/IP vendors are currently establishing a standard TCP/IP application program interface (API) called the Windows Sockets (or *WinSock*) API. At the moment however, the WinSock API has not been universally adopted, and implementations of WinSock are inconsistent between vendors.

The APP2SOCK ("application to socket") specification insulates you through the transition from the multiple APIs existing within your organization to a mature WinSock API standard. Your application investments will be secure with existing technology as well as the WinSock standard. Supported third-party transports include Novell, Microsoft, FTP, SunSelect, NetManage, Frontier, Wollongong, Distinct, WRQ, and Beame & Whiteside.

A Single Interface to Desktop TCP/IP

Implementing APP2SOCK provides you a single interface to multiple API's and transports. The APP2SOCK API transparently functions as a "shock absorber" supporting both TSR and DLL-based TCP/IP transports.

As you incorporate this innovative architecture into your environment, you will benefit from SPRY's expertise in integrating multiple API's on the desktop. SPRY is committed to continued development and expanded support of multiple protocols by APP2SOCK. APP2SOCK will incorporate support for Windows NT, IPX/SPX, DECnet and NetBEUI transport protocols.

The APP2SOCK Developer's Kit

The APP2SOCK specification is now available for Windows application development. APP2SOCK features a concise set of easy to implement function calls which will allow developers to write their applications only once, and still be able to support multiple socket interfaces.

Future development of the APP2SOCK API will include these implementations:

- More error codes will be included.
- CommAllowNewServer will support multiple simultaneous connections.
- Support will be added for identifying and returning the underlying socket library and socket library version number.
- A callback function will eliminate the need to poll for new incoming data.
- Some functions will be combined, reducing the number of calls required for an operation.

What's Included with the APP2SOCK Developer's Kit

APP2SOCK Files:

- APP2SOCK.LIB
- APP2SOCK.DLL
- APP2SOCK.H

The APP2SOCK Technical Guide:

- Documentation on APP2SOCK function calls
- Sample APP2SOCK code

APP2SOCK Functions

The APP2SOCK Functions are listed below. This guide provides a synopsis, description, and return values for each function.

COMMInitial	Initialize the socket environment.
COMMListen	Open a APP2SOCK session to listen to a connection request.
COMMOpen	Open a APP2SOCK session to request a connection.
COMMTestConnected	Check on the connect status of an APP2SOCK session.
COMMAllowNewServer	Accept a new request for a server connection.
COMMAllowEvent	Check on the incoming data for an APP2SOCK session.
COMMCharsPresent	Determine the number of unread bytes in a APP2SOCK session.
COMMRead	Read a specified number of characters from the network.
COMMPeek	Peek at the incoming or unread characters.
COMMWriteCharacter	Transmit a single byte.
COMMWriteBlock	Transmit a string of specified length.
COMMWriteASCIIString	Transmit a string which may contain encoded control characters.
COMMClose	Close a APP2SOCK session connection and the associated socket.
COMMGetPeerInfo	Retrieve the remote IP address, remote port, and local port into an integer array.
COMMGetMyIP	Get local PC's IP address.
COMMErrorMessageString	Give an error message.
COMMFakeOpen	Open a APP2SOCK session for a previously opened socket.
COMMFakeClose	Close a APP2SOCK session which was opened by COMMFakeOpen.
COMMWaitForData	Check incoming data in a given period of time.

COMMInitial()

Synopsis:

```
#include <app2sock.h>
bLoad = COMMInitial()
BOOL        bLoad;
```

Description:

APP2SOCK needs to talk to various socket libraries. Some socket libraries request that user application programs call their initialization functions once, but some request this more than once. COMMInitial() will make sure that the right initial functions are being called the correct number of times.

COMMInitial() is the first function that needs to be called by an application program to initialize the underlying functions.

Return Value:

BOOL	TRUE	initialize success
	FALSE	initialize failure

COMMListen()

Synopsis:

```
#include <app2sock.h>

a2sID = COMMListen(uSap, lpintsocket)

A2SID             a2sID;

unsigned int      uSap;

LPINT             lpintsocket;
```

Description:

COMMListen() opens server-type socket operations. The server-type socket will listen to the connection requirement from other machines' client programs. The COMMAllowNewServer function allows you to check new connection requirements.

The parameter **uSap** gives you the service access port (SAP), which is the service the server socket will provide for peer applications. For example, if you want to provide a FTP server function, the uSap should be 21. The **lpintsocket** will tell you the socket id for the listening socket you open. Normally, you won't need to know **lpintsocket**; it is provided for advanced functions.

Return Value:

int	positive or 0	APP2SOCK ID, which is used to reference the socket you opened.	
	negative	failure	
		-1	generate failure
		-2	out of resources
		-3	get local pc ip name failed
		-4	get remote host information failed
		-5	no socket available
		-6	I/O control failure
		-7	binding failure
		-8	listen to incoming packet failure

COMMOpen()

Synopsis:

```
#include <app2sock.h>

a2sID = COMMOpen(lpszHost, iflag, wSap, wMyAp,
sMode, lpintsocket)

A2SID           a2sID;
LPSTR           lpszHost;
int             iflag;
WORD            wSap, wMyAp;
short far *     sMode;
LPINT           lpintsocket;
```

Description:

COMMOpen() opens a client-type socket to start a connection request. COMMAllowEvent() allows you to check incoming data. Since there are many different socket libraries, some of them will need more time to establish a connection. There is a parameter in APP2SOCK.INI that allows a user to set up the open delay time, so that this function will delay a certain number of seconds before socket open success. The setting is as follows:

[APP2SOCK]
OpenDelay=1 # 1 second

This function takes 6 parameters: **lpszHost** is a LPSTR to either host name or IP address in dotted notation; **iflag** should be either BYIP for a dotted-notation IP address or BYNAME for hostname; **wSap** is the service access port that you try to connect to; **wMyAp** is the service access port number, (this should be 0 when you are writing a client application and do not need to worry about your own SAP); **sMode** doesn't take an initial value, it is a

return of socket mode. (Currently when opening a client socket, we only support synchronous nonblocking mode; this return value is not changeable); **lpintsocket** is the opened socket id for this function—this is for advanced use only.

Return Value:

int	positive or 0	APP2SOCK ID, which is used to reference the socket you opened.
	negative	failure
		-1 generate failure
		-2 out of resources
		-3 get local pc ip name failed
		-4 get remote host information failed
		-5 no socket available
		-6 I/O control failure
		-7 binding failure
		-8 listen to incoming packet failure
		-9 get remote ip address failure
		-10 connect failure or timeout

COMMTestConnected()

Synopsis

```
#include <app2sock.h>

bConnect = COMMTestConnected(a2sID);

BOOL        bConnect;
A2SID       a2sID;
```

Description:

COMMTestConnected() returns the status of connection for the a2sID. You must first call COMMAllowEvent(); COMMTestConnected() will then return True or False based on the result of COMMAllowEvent().

Return Value:

BOOL	TRUE	connected.
	FALSE	not connected.

COMMAllowNewServer()

Synopsis:

```
#include <app2sock.h>

sid = COMMAllowNewServer(a2sID);

int         sid;
A2SID       a2sID;
```

Description:

COMMAllowNewServer() checks whether other machines want to make a connection to the local PC. If so, it will open a socket for the connection, and close the current listening socket. The new opened socket ID will be returned by this function.

Return Value:

int	> 0	new socket id
	= 0	no new server event
	-11	accept server function failure

COMMAllowEvent()

Synopsis:

```
#include <app2sock.h>

istatus = COMMAllowEvent(a2sID);

int         istatus;
A2SID       a2sID;
```

Description:

COMMAllowEvent() will check whether there is new incoming data for the a2sID session which was opened by COMMOpen(). This is a pooling function. The status of this a2sID session will not be changed until the application calls this function.

Return Value:

int	-1	a2sID is not available
	0	there is no new incoming packet for a2sID
	-12	Receive would block
	2	the first time an incoming packet is received
	3	there is a new incoming packet (the port is connected).

COMMCharsPresent()

Synopsis:

```
#include <app2sock.h>

inum = COMMCharsPresent(a2sID)

int          inum;
A2SID        a2sID;
```

Description:

COMMCharsPresent() tells you how many unread bytes of data are still in the socket buffer. However, this function will not check the incoming packet for you. It only returns the value from the previous COMMAllowEvent() call. Use COMMPeek() to check the incoming data.

Return Value:

short how many bytes of unread (pending) data are in the queue.

COMMRead()

Synopsis:

```
#include <app2sock.h>

inum = COMMRead(a2sID, lpszBuf, icount)

int         inum;
A2SID       a2sID;
LPSTR       lpszBuf;
short       icount;
```

Description:

COMMRead() tries to read **icount** characters from the socket buffer. If there are less than **icount** characters to read, COMMRead() will read as many as possible. The read in data will be stored in **lpszBuf**.

Return Value:

int	-1	connection is closed.
	0	no data to read
	-12	receiving data would block
	> 0	how many bytes of data were read in.

COMMPeek()

Synopsis:

```
#include <app2sock.h>

inum = COMMPeek(a2sID, lpszBuf, icount)

int         inum;
A2SID       a2sID;
LPSTR       lpszBuf;
short       icount;
```

Description:

COMMPeek() checks how many unread bytes of incoming data are in the buffer for the a2sID session, up to **icount** bytes.

Return Value:

int	0	no incoming data
	-12	Receiving data would block
	> 0	How much data is in buffer

COMMWriteCharacter()

Synopsis:

```
#include <app2sock.h>

inum = COMMWriteCharacter(a2sID, wOutByte)

int          inum;
A2SID        a2sID;
WORD         wOutByte;
```

Description:

COMMWriteCharacter() sends out a byte to the socket library. Even if you take the parameter **wOutByte** as a word, only the lower byte of **wOutByte** will be sent out.

Return Value:

int	-1	general failure
	0	this application port is not connected
	1	the outgoing byte is sent out successfully

COMMWriteBlock()

Synopsis:

```
#include <app2sock.h>

inum = COMMWriteBlock(a2sID, lpszBuf, icount)

int         inum;
A2SID       a2sID;
LPSTR       lpszBuf;
int         icount;
```

Description:

COMMWriteBlock() sends out **icount** bytes of the characters pointed by **lpszBuf**. This function will return how many bytes were sent to the socket. It is possible that only partial bytes are being sent, and the return value **inum** will tell you how many bytes are being sent successfully.

Return Value:

int	-1	general failure
	0	not connected
	> 0	bytes of data sent out

COMMWriteASCIIString()

Synopsis:

```
#include <app2sock.h>

bSent = COMMWriteASCIIString(a2sID, lpBuf, icount)

BOOL        bSent;
A2SID       a2sID;
LPSTR       lpBuf;
int         icount;
```

Description:

COMMWriteASCIIString() transmits **icount** bytes of the characters pointed to by **lpszBuf**. This function is similar to COMMWriteBlock(), and it can transmit encoded control characters, which COMMWriteBlock() can't.

Return Value:

BOOL	FALSE	a2sID is invalid, or transmit failure.
	TRUE	buffer sent out successfully.

COMMClose()

Synopsis:

```
#include <app2sock.h>

COMMClose(a2sID);

A2SID       a2sID;
```

Description:

COMMClose() will drop the a2sID session connection, and close the socket that a2sID is associated with. There is no return value for this function call.

Return Value:

VOID

COMMGetPeerInfo()

Synopsis:

```
#include <app2sock.h>

COMMGetPeerInfo(lpInfo, a2sID)

LPINT       lpInfo;
A2SID       a2sID;
```

Description:

COMMGetPeerInfo() returns an array of integers which are pointed to by **LPINT lpInfo**. The **lpInfo** is in the following structure:

lpInfo[0] - The first integer in the dotted notation IP address.
lpInfo[1] - The second integer in the dotted notation IP address.
lpInfo[2] - The third integer in the dotted notation IP address.
lpInfo[3] - The fourth integer in the dotted notation IP address.
lpInfo[4] - The high word of the remote Service Access Port (SAP) in network order.
lpInfo[5] - The low word of the remote SAP (network order).
lpInfo[6] - The high word of the local SAP (network order).
lpInfo[7] - The low word of the local SAP (network order).

Return Value:

VOID

COMMGetMyIP()

Synopsis

```
#include <app2sock.h>

COMMGetMyIP(lpMyIP, a2sID)

BYTE far *  lpMyIP;
A2SID       a2sID;
```

Description:

COMMGetMyIP() returns the local PC's IP address from the first parameter **lpMyIP**. The **lpMyIP**'s structure is as follows:

lpMyIP[0] - The first integer in the dotted notation IP address.
lpMyIP[1] - The second integer in the dotted notation IP address.
lpMyIP[2] - The third integer in the dotted notation IP address.
lpMyIP[3] - The fourth integer in the dotted notation IP address.

Return Value:

VOID

COMMGetErrorMessageString()

Synopsis:

```
#include <app2sock.h>

lpszErrMsg = COMMGetErrorMessageString(ierrno)

LPSTR       lpszErrMsg;
int         ierrno;
```

Description:

COMMGetErrorMessageString() returns a character string of error message. The **ierrno** is the number that was received from the previous function call.

Return Value:

LPSTR	a long pointer string to a corresponding error message.

COMMFakeOpen()

Synopsis:

```
#include <app2sock.h>

a2sID = COMMFakeOpen(socketid)

A2SID       a2sID;
int         socketid;
```

Description:

COMMFakeOpen() does not open a new socket. It takes **socketid**, which may be opened by another application, and creates an a2sID session for this opened socketid. The status of this a2sID will be set to CONNECTED by default.

Return Value:

int	-2	no more sessions are available
	>= 0	port number for this session

COMMFakeClose()

Synopsis:

```
#include <app2sock.h>

COMMFakeClose(a2sID)

A2SID       a2sID;
```

Description:

COMMFakeClose() closes the a2sID session which was opened by COMMFakeOpen(). This function only closes the a2sID session, but not the opened socket.

Return Value:

VOID

COMMWaitForData()

Synopsis:

```
#include <app2sock.h>

bData = COMMWaitForData(a2sID, lTimeout)

BOOL        bData;
A2SID       a2sID;
long        lTimeout;
```

Description:

COMMWaitForData() checks whether the a2sID session has incoming data in a certain period of time (**lTimeout**). If there is incoming data, this function will return immediately, otherwise, it will keep checking new data for **lTimeout** seconds. COMMWaitForData() will not read data, it will only check for incoming data. If the return value is TRUE (this a2sID session has new incoming data), you need to use the function COMMRead() to read the incoming data. COMMWaitForData() will translate and dispatch Windows Messages, so other Windows applications can still process during the waiting period.

Return Value:

BOOL	TRUE	There is some data for this A2SID
	FALSE	No data is here

Sample APP2SOCK Code

This is a Windows C sample program which uses Timer to read in incoming information. The APP2SOCK functions used are highlighted as bold.

```
/************************************************************************

        PROGRAM: Sample.c

        PURPOSE: A sample program for APP2SOCK
************************************************************************/

#include <windows.h>            /* required for all Windows applications */
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <app2sock.h>

HANDLE hInst;                   /* current instance */
int pnum = -1;
HWND hwndMain;

/
************************************************************************

        FUNCTION: WinMain(HANDLE, HANDLE, LPSTR, int)

************************************************************************/

int PASCAL WinMain(hInstance, hPrevInstance, lpCmdLine, nCmdShow)
        HANDLE hInstance;           /* current instance */
        HANDLE hPrevInstance;       /* previous instance */
        LPSTR lpCmdLine;            /* command line */
        int nCmdShow;               /* show-window type (open/icon) */
{
        MSG msg;

        if (!hPrevInstance)         /* Other instances of app running? */
```

```
                if (!InitApplication(hInstance))        /* Initialize shared things */
                        return (FALSE);                 /* Exits if unable to initialize */

        if (!InitInstance(hInstance, nCmdShow))
                return (FALSE);

        /* Acquire and dispatch messages until a WM_QUIT message is received. */

        while (GetMessage(&msg,   NULL,  NULL, NULL) )
        {
                if ( hCommandDlg==NULL || !IsDialogMessage(hCommandDlg, &msg) )
                {
                        TranslateMessage(&msg);
                        DispatchMessage(&msg);
                }
        }
        return (msg.wParam);        /* Returns the value from PostQuitMessage */
}

/*************************************************************************

        FUNCTION: InitApplication(HANDLE)

*************************************************************************/

BOOL InitApplication(hInstance)
        HANDLE hInstance;                       /* current instance */
{
        WNDCLASS      wc;

        wc.style = NULL;                        /* Class style(s). */
        wc.lpfnWndProc = MainWndProc;           /* Function to retrieve messages for */
        wc.cbClsExtra = 0;                      /* No per-class extra data. */
        wc.cbWndExtra = 0;                      /* No per-window extra data. */
        wc.hInstance = hInstance;               /* Application that owns the class. */
        wc.hIcon = LoadIcon(hInstance, "COMMTEST");
        wc.hCursor = LoadCursor(NULL, IDC_ARROW);
        wc.hbrBackground = GetStockObject(WHITE_BRUSH);
```

```
        wc.lpszMenuName =  "SampleMenu";        /* Name of menu resource in .RC file. */
        wc.lpszClassName = "SampleClass";       /* Name used in call to CreateWindow. */

        return (RegisterClass(&wc));

}

/*******************************************************************************
        FUNCTION:  InitInstance(HANDLE, int)

*******************************************************************************/

BOOL InitInstance(hInstance, nCmdShow)
        HANDLE          hInstance;              /* Current instance identifier.       */
        int             nCmdShow;               /* Param for first ShowWindow() call. */
{
        HWND            hWnd;                   /* Main window handle.                */
        HWND            hReply;

        hInst = hInstance;

        hWnd = CreateWindow(
                "SampleClass",                  /* See RegisterClass() call. */
                "Communication Test",           /* Text for window title bar. */
                WS_OVERLAPPEDWINDOW,            /* Window style. */
                CW_USEDEFAULT,                  /* Default horizontal position. */
                CW_USEDEFAULT,                  /* Default vertical position. */
                480,                            /* Default width. */
                420,                            /* Default height */
                NULL,                           /* Overlapped windows have no parent. */
                NULL,                           /* Use the window class menu. */
                hInstance,                      /* This instance owns this window. */
                NULL                            /* Pointer not needed. */
        );

        if (!hWnd)
                return (FALSE);

        // start timer
```

```
        if (!SetTimer(hWnd, 1, 500, NULL))
        {
                MessageBox(hWnd, "Too many timers!", "WARNING",
                MB_OK|MB_ICONSTOP);
                return FALSE;
        }

        ShowWindow(hWnd, nCmdShow);/* Show the window */
        UpdateWindow(hWnd);             /* Sends WM_PAINT message */
        return (TRUE);                  /* Returns the value from PostQuitMessage */

}

/*************************************************************************

        FUNCTION: MainWndProc(HWND, UINT, WPARAM, LPARAM)

*************************************************************************/

long CALLBACK __export MainWndProc(hWnd, message, wParam, lParam)
        HWND hWnd;                      /* window handle */
        UINT message;                   /* type of message */
        WPARAM wParam;                  /* additional information */
        LPARAM lParam;                  /* additional information */
{

        switch (message)
        {
                case WM_CREATE:
                        COMMInitial();          /*APP2SOCK Initialization command */
                        return FALSE;

                case WM_COMMAND:                /* message: command from application */
                                                /* menu */
                        switch (wParam)
                        {
                                case IDM_OPENCONNECTION:
                                {
                                        int socketmode, sd;
```

```
            /* Open APP2SOCK connection to      */
            /* IP address (Telnet session, port 23) */

            pnum=COMMOpen("165.121.3.3",
                        BYIP,
                        23,
                        0,
                        &socketmode,
                        &sd);

            if ((pnum <= QERROR)    /* NETWORK ERROR */
            {           pnum = -1;
                        MessageBox(hDlg, "Can not open
                        APP2SOCK Session!", "Sample",
                        MB_OK|MB_ICONSTOP);
            }

            break;
}

case IDM_CLOSECONNECTION:
            if (pnum >= 0)


            /* Close the open APP2SOCK session */

            {           COMMClose(pnum);
                        pnum = -1;
            }

            break;

case IDM_SENDBLOCKDATA:
{
            char szbuf[64];    // Assume your data is in this buffer
                               // already
```

```
                        int i;



                        /* Send the szbuf to the open a2sID session
                          (pnum) */

                        i = COMMWriteBlock(pnum, szbuf,
                        strlen(szbuf));
                        if (i  == strlen(szbuf))
                                MessageBox(hWnd, szbuf,
                                        "COMMWriteBlock() success",
                                        MB_OK);
                        break;
                }
                default:                /* Lets Windows process it */
                        return (DefWindowProc(hWnd, message, wParam,
                        lParam));
        }
        break;

case WM_TIMER:
{       char buf[164];
        /* If there is incoming data, read it in */

        if (COMMAllowEvent(pnum) == 3)
        {
                memset(buf, 0, bufsize);
                while (COMMRead(pnum, buf, bufsize-1) > 0)
                {
                        MessageBox(hWnd,buf, "COMMRead", MB_OK);
                        memset(buf, 0, bufsize);
                }
        }
        break;
}

case WM_CLOSE:
        /* Close a2sID session */
```

```
                    COMMClose(pnum);
                    DestroyWindow(hWnd);
                    break;

                case WM_DESTROY:            /* message: window being destroyed */
                    KillTimer(hWnd, 1);
                    DeleteObject(hFont);
                    PostQuitMessage(0);
                    break;




                default:                    /* Passes it on if unprocessed    */
                    return (DefWindowProc(hWnd, message, wParam, lParam));
}
return (NULL);
}
```