Appendix B: NET_CON Source Code

NET_CON.CPP

```
/*
 *                  LSA1000 Sample Network Connection
 *
 *                  Copyright (c) 1998 LeCroy Corporation
 *
 *                  Written By: Ricardo Palacio
 *
 *                  April, 1998
 *
 *
 */


/*
 * $Header:$
 * $Log:$
 *
 */


#include <windows.h>
#include <winsock.h>
#include <stdio.h>
#include <time.h>
#include "net_con.h"


static int                      hSocket;
static int                      sTimeout = 3;
static int                      sWinsockInitFlag = FALSE;
static char                     sCurrentAddress[256];
static int                      sConnectedFlag = FALSE;


#define CMD_BUF_LEN         8192
static char                 sCommandBuffer[CMD_BUF_LEN];



int TCP_Connect(char *ip_address)
{
SOCKADDR_IN   serverAddr;
```

```
int sockAddrSize = sizeof (SOCKADDR), result;
WORD wVersionRequested;
WSADATA wsaData;
const int resp = 1;
char tmpStr[512];
fd_set wr_set = {1, {0}};
TIMEVAL tval;
unsigned long argp;



       if (sConnectedFlag==TRUE)
              return -1;

       strcpy(sCurrentAddress, ip_address);

       tval.tv_sec = sTimeout;
       tval.tv_usec = 0;



       if (!sWinsockInitFlag)
       {
              wVersionRequested = MAKEWORD(1, 1);
              if (WSAStartup(wVersionRequested, &wsaData) != 0)
              {
MessageBox(0, "Unable to initialize the Windows socket environment.", "ERROR",
MB_OK);
                     return -1;
              }
              sWinsockInitFlag = TRUE;
       }

    /* build server socket address */
       serverAddr.sin_family = AF_INET;
       serverAddr.sin_port = htons (SERVER_PORT);

       if ((serverAddr.sin_addr.s_addr = inet_addr(ip_address)) == -1)
       {
           MessageBox(0, "Bad server address", "ERROR", MB_OK);
           return -1;
       }

    /* create client's socket */
```

```
if ((hSocket = socket(AF_INET, SOCK_STREAM, 0)) == INVALID_SOCKET)
       {
MessageBox(0, "Unable to create client's socket.", "ERROR", MB_OK);
           return -1;
       }

    if (setsockopt(hSocket, IPPROTO_TCP, TCP_NODELAY, (char*)&resp, sizeof(resp))
!= 0)
       {
              MessageBox(0, "Unable to set socket option to TCP_NODELAY", "ERROR",
MB_OK);
              return -1;
       }


       wr_set.fd_array[0] = hSocket;

       argp = 1;//non blocking mode
       ioctlsocket(hSocket, FIONBIO, &argp);

       connect(hSocket, (SOCKADDR FAR *) &serverAddr, sockAddrSize);

       result = select(hSocket, NULL, &wr_set, NULL, &tval);

       argp = 0;//blocking mode
       ioctlsocket(hSocket, FIONBIO, &argp);

       /* connect to server (scope) */
       if (result < 1)
       {
           sprintf(tmpStr, "Unable to make connection to IP:%s", ip_address);
              MessageBox(0, tmpStr, "TIMEOUT", MB_OK);
           return -1;
       }

       sConnectedFlag = TRUE;

       return 0;
}
```

```
int TCP_Disconnect(void)
{

if (sConnectedFlag != TRUE)
              return -1;

closesocket(hSocket);
sConnectedFlag = FALSE;

return 0;
}
```

```
int TCP_WriteDevice(char *buf, int len, BOOL eoi_flag)
{

       TCP_HEADER header;
       int result, bytes_more, bytes_xferd;
       char *idxPtr;



       if (sConnectedFlag != TRUE)
              return -1;

       if (len < CMD_BUF_LEN)
              strcpy(sCommandBuffer, buf);

       // set the header info
       header.bEOI_Flag = DATA_FLAG;
       header.bEOI_Flag |= (eoi_flag)? EOI_FLAG:0;
       header.reserved[0] = 1;
       header.reserved[1] = 0;
       header.reserved[2] = 0;
       header.iLength = htonl(len);


       // write the header first
```

```
	if (send(hSocket, (char *) &header, sizeof(TCP_HEADER), 0) !=
sizeof(TCP_HEADER))
	{
	    MessageBox(0, "Unable to send header info to the server.", "ERROR",
MB_OK);
	    return -1;
	}


	bytes_more = len;
	idxPtr = buf;
	bytes_xferd = 0;
	while (1)
	{
		// then write the rest of the block
		idxPtr = buf + bytes_xferd;


		if ((result = send(hSocket, (char *) idxPtr, bytes_more, 0)) < 0)
		{
			MessageBox(0, "Unable to send data to the server.", "ERROR",
MB_OK);
			return -1;
		}

		bytes_xferd += result;
		bytes_more -= result;
		if (bytes_more <= 0)
			break;
	}

	return 0;
}




int TCP_ClearDevice(void)
{

	if (sConnectedFlag != TRUE)
		return -1;
```

```
        TCP_Disconnect();
        TCP_Connect(sCurrentAddress);
        return 0;
}
```

```
int TCP_ReadDevice(char *buf, int len, int *recv_count)
{
        TCP_HEADER header;
        char tmpStr[512];
        int result, accum, space_left, bytes_more, buf_count;
        char *idxPtr;
        fd_set rd_set = {1, {0}};
        TIMEVAL tval;



        if (sConnectedFlag != TRUE)
                return -1;

        *recv_count = 0;

        if (buf==NULL)
                return -1;

        rd_set.fd_array[0] = hSocket;
        tval.tv_sec = sTimeout;
        tval.tv_usec = 0;



        memset(buf, 0, len);
        buf_count = 0;
        space_left = len;

        while (1)
        {
                // block here until data is received of timeout expires
```

```
		result = select(hSocket, &rd_set, NULL, NULL, &tval);
		if (result < 1)
		{
			TCP_ClearDevice();
			MessageBox(0, sCommandBuffer, "Read timeout", MB_OK);
			return -1;
		}
		// get the header info first
		accum = 0;
		while (1)
		{
			memset(&header, 0, sizeof(TCP_HEADER));

if ((result = recv(hSocket, (char *) &header + accum, sizeof(header) - accum, 0))
< 0)
			{
				TCP_ClearDevice();
MessageBox(0, "Unable to receive header info from the server.", "ERROR", MB_OK);
				return -1;
			}

			accum += result;
			if (accum>=sizeof(header))
				break;
		}

		header.iLength = ntohl(header.iLength);
		if (header.iLength < 1)
			return 0;

		// only read to len amount
		if (header.iLength > space_left)
		{
			header.iLength = space_left;
			sprintf(tmpStr, "Read buffer size (%d bytes) is too small",
len);
			MessageBox(0, tmpStr, "ERROR", MB_OK);
		}

		// read the rest of the block
		accum = 0;
		while (1)
		{
			idxPtr = buf + (buf_count + accum);
			bytes_more = header.iLength - accum;
			if ((space_left-accum) < TCP_MINIMUM_PACKET_SIZE)
```

```
                        {
                                TCP_ClearDevice();
sprintf(tmpStr, "Read buffer needs to be adjusted, must be minimum of %d bytes",
TCP_MINIMUM_PACKET_SIZE);
                                MessageBox(0, tmpStr, "ERROR", MB_OK);
                                return -1;
                        }

if ((result = recv(hSocket, (char *) idxPtr, (bytes_more>2048)?2048:bytes_more,
0)) < 0)
                        {
                                TCP_ClearDevice();
MessageBox(0, "Unable to receive data from the server.", "ERROR", MB_OK);
                                return -1;
                        }


                        accum += result;
                        if (accum >= header.iLength)
                                break;
                        if ((accum + buf_count) >= len)
                                break;
                }
                buf_count += accum;
                space_left -= accum;

                if (header.bEOI_Flag & EOI_FLAG)
                        break;
                if (space_left <= 0)
                        break;
        }

        *recv_count = buf_count;

        return 0;
}




int TCP_SetTimeout(int seconds)
{
```

```
       sTimeout = seconds;
       return 0;
}




int main(int argc, char *argv[])
{
    char replyBuf[512];
       int read;


       if (argc < 2)
       {
              printf("\nEXAMPLE: net_con 172.28.11.22\n");
              return 0;
       }


    if (TCP_Connect(argv[1]))
              return 0;


       if (TCP_WriteDevice("*idn?\n", 6, TRUE))
       {
              TCP_Disconnect();
              return 0;
       }


       if (TCP_ReadDevice(replyBuf, 512, &read))
       {
              TCP_Disconnect();
              return 0;
       }

       if (TCP_Disconnect())
              return 0;

    printf("Scope's reply: %s\n", replyBuf);

    return 0;
}
```

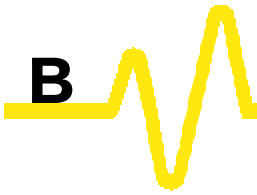
```
                                #define SERVER_PORT             1861

#define EOI_FLAG                0x01
#define SRQ_FLAG                0x08
#define CLEAR_FLAG              0x10
#define LOCKOUT_FLAG            0x20
#define REMOTE_FLAG             0x40
#define DATA_FLAG               0x80

#define READ_TIME_OUT                   10

#define TCP_MINIMUM_PACKET_SIZE   64

typedef struct
{
       unsigned char        bEOI_Flag;
       unsigned char reserved[3];
       int           iLength;
} TCP_HEADER;
```



Appendix B: NET_CON Source Code