LEP USB Programming Manual

Purpose:

This document describes the LEP USB interface and how to program it.

Requirements:

- LEP USB Controller
- LEP USB Driver
- USB ready computer running Win98 or Win 2000
- Microsoft Visual C++ version 4.0 or better, or equivalent 32 bit compiler for x86 based system, or Visual basic
- Knowledge of C or Visual basic

Installation:

To install the LEP driver, attach the USB cable to both your computer and the LEP USB controller and power up. The computer should automatically recognize the new hardware. Click the browse button and point to the disk containing the LEP driver software. The driver will be automatically install. You can test the USB interface by running any one of the example programs LEPUSB.exe, Usb_rCfg.exe or Usb_VB.exe. You can find these files on the driver distribution disk or on the LEP web site at www.ludl.com.

Overview:

To communicate with LEP USB Driver one must first enumerate the device. The enumeration of the device returns a device name. This device name is used to open the interface, using CreateFile(). Once you have the handle from CreateFile() you can use ReadFile(), WriteFile(), DeviceIOControl(), and CloseHandle() to communicate to the LEP USB Controller. The hardest part is getting the device name the rest is simply. Example code in console C, Windows Visual Basic, and Windows MFC C++ can be found in appendix B,C, and D respectively.

 Ludl Electronic Products Ltd. ©2000

LEP USB Programming Manual

Device Enumeration:

The device name consists of a number representing a physical port plus the GUID (global unique identifier) appended to it. The GUID for the LEP device is {4d48f140-44e2-11d3-9d64-00e0291dee58}, and a typical device name looks like \\.\0000000000000002#{4d48f140-44e2-11d3-9d64-0e0291dee58}. The device name changes each time you plug in an additional device or plug the device into a different USB port or hub.

There are two ways to get the device name. The easiest method is to read the device name from the registry. When a LEP USB device is plugged in to your computer, the OS detects it and loads the driver. When this driver loads the device name is stored in the registry. The user just has to read the name out of the registry. This method has one disadvantage. It can't be used when more than one USB LEP device is plugged in to your computer, because only the last device name will recorded in the registry.

To use the registry method simply open the registry key, query the value, and close the registry key. The registry key name is `DeviceName` and the path is HKEY_LOCAL_MACHINE\System\CurrentControlSet\Services\LepUsb\Parameters\ .
Here's an example is Visual Basic.

```
DIM DeviceName as string
DeviceName = GetRegValue(HKEY_LOCAL_MACHINE, _
                         "System\CurrentControlSet\Services\LepUsb\Parameters\", _
                         "DeviceName")


' GetRegValue - Gets the Key value in the registry given a registry key.
Function GetRegValue(hKey As Long, lpszSubKey As String, szKey As String ) As String

Dim phkResult As Long, lResult As Long
Dim szBuffer As String, lBuffSize As Long

'Create Buffer
szBuffer = Space(255)           ' Allocate buffer space
lBuffSize = Len(szBuffer)       ' Set the length

RegOpenKeyEx hKey, lpszSubKey, 0, 1, phkResult      'Open the Key, get a handle to it

lResult = RegQueryValueEx(phkResult, szKey, 0, 0, szBuffer, lBuffSize) 'Query the value

RegCloseKey phkResult           'Close the Key

If lResult = ERROR_SUCCESS Then
    GetRegValue = szBuffer           ' return key value
End If
Exit Function
```

The second method to get the device name is two use the device manger. To do this one calls functions in the setupapi DLL. Simply poll the device manger with the LEP GUID for all the device that match the GUID given. The device manger will return the device names for all the devices currently available.
Below is a C example using this method.

Use the DEFINE_GUID macro to build the GUID.

```
DEFINE_GUID(GUID_LEP, 0x4d48f140, 0x11d3, 0x9d, 0x64, 0x00,
      0xe0, 0x29, 0x1d, 0xee, 0x58);
```

This GUID is passed to SetupDiGetClassDevs(), of which returns a handle to the device. The enumeration functions are found in the setupapi library.

```
HDEVINFO hinfo = SetupDiGetClassDevs(&GUID_LEP, NULL, NULL,
      DIGCF_PRESENT | DIGCF_INTERFACEDEVICE);
```

The first argument identifies the interface you're looking for. The flag bits in the last argument indicate that you are looking for the interfaces exported by the LEP USB device.

Once you have a handle to the device information set, you can perform a enumeration of all the devices that export the particular interface you're interested in. See Microsoft function documentation for more information on setupapi library functions.

Poll the device manager till there are no matching devices left.

```
   int i;
   Cstring Devices[10];              // an array of cstrings
   for (DWORD i=0; ; ++i)
      {
      SP_INTERFACE_DEVICE_DATA Interface_Info;
      Interface_Info.cbSize = sizeof(Interface_Info);
      // Enumerate device
      if (!SetupDiEnumInterfaceDevice(hInfo, NULL, (LPGUID)
                &LEP_GUID,i, &Interface_Info))
         {
         SetupDiDestroyDeviceInfoList(hInfo);
         return(i);
         }

      DWORD needed;                        // get the required lenght
      SetupDiGetInterfaceDeviceDetail(hInfo, &Interface_Info,
                                 NULL, 0, &needed, NULL);
      PSP_INTERFACE_DEVICE_DETAIL_DATA detail =
            (PSP_INTERFACE_DEVICE_DETAIL_DATA) malloc(needed);
      if (!detail)
         {
         SetupDiDestroyDeviceInfoList(hInfo);
         return(i);
         }
                                 // fill the device details
      detail->cbSize = sizeof(SP_INTERFACE_DEVICE_DETAIL_DATA);
      if (!SetupDiGetInterfaceDeviceDetail(hInfo, &Interface_Info,
                                 detail, needed,NULL, NULL))
         {
         free((PVOID) detail);
         SetupDiDestroyDeviceInfoList(hInfo);
         return(i);
         }

      char name[MAX_PATH];
      strncpy(name, detail->DevicePath, sizeof(name));
      free((PVOID) detail);
      Devices[i] = name;      // keep a copy of each device name
      }  // end of for loop
```

After this code runs you end up with a list of device names, or NULL if no devices could be found (i = 0). Each device name will represent one LEP USB controller that is plugged into your computer. If you know that you will only support one LEP USB controller on your system at one time, you can reduce the enumeration code by dropping the for loop and only going through the code once. The device name(s) that are returned from the above code has a port number prefixed to the original GUID. The port number is related to order of the plug and play device on your machine and can not be predetermined. The device name should look like the following.

```
\\.\0000000000000002#{4d48f140-44e2-11d3-9d64-00e0291dee58}
```

This is the complete device name we need to communicate with the LEP USB device.

LEP USB Programming Manual

Device Communications:

Open Device:

To begin communicating with the LEP USB Controller you must first acquire a handle to it. To do this just pass the device name to the CreateFile() function. This is done in the same manner as opening or creating a file. If successful this function will return a handle to the device. If the device is not plugged in, unpowered, or opened by another program this function will fail.

```
HANDLE hUsbDevice = CreateFile(devicename, GENERIC_READ |
            GENERIC_WRITE, 0, NULL, OPEN_EXISTING, 0, NULL);
```

Reading and Writing:

Once the handle is acquired you may read and write to the LEP USB device. To read call the ReadFile() function.

```
ReadFile(hUsbDevice, pText, nBytes, &nBytesRead, NULL);
```

The ReadFile() function will try to read the number of byte asked for and return the request in the pointer passed. If the number of bytes requested is larger than what the driver IN buffer has, it will wait till either; the data comes or till it times out. The default timeout is 30 seconds. The driver IN buffer is defaulted to 512 bytes. If this buffer fills up, USB IN packet will be halted till the buffer empties.

To write to the LEP USB device use the WriteFile() function.

```
WriteFile(hUsbDevice, pText, strlen(pText), &nBytesWritten, NULL);
```

The WriteFile() function will write the data passes to the LEP USB device. The function will return when either all the data has been written or an error has occurred. The driver OUT buffer size is created dynamically so any buffer size may be passed.

Close Communication:

When your application has finished using the device, the device should be closed. To do this call CloseHandle() with the device handle. If you do not close the device, you will not be able to access it again without re-setting the device port.

```
CloseHandle( hUsbDevice ) ;
```

Device I/O Controls:

The DeviceIOControl() function allow the application program to read driver status and set driver variable. This function accepts CTL_CODES that are passed to the device driver. Each code represents a function. The following available functions are described below.

Receive Data Length: - `IOCTL_LEPUSB_GET_READ_LENGTH`

In order to make device reads with out waiting for a time out condition, the application program needs to know how much data exist in the driver's receive buffer. To do this, use the DeviceIoControl function to issue a read IN buffer length command. First construct and fill the ReadLenStruct data structure and call DeviceIOControl() with the data structure as the OUT buffer. When the function returns the data structure is filled with all the IN buffer length data. The data structure returns the following; The length of the currently available data in the buffer. The available free space in the buffer. The length of data up to and including the first line feed (LF) character found. Lastly the length of the data up to and including the first match of the user character passed. If the LF or user character are not found a zero length is returned. To set the user defined character, pass it as the first character in the IN buffer portion of the DeciveIoControl function. See the function definition below.

```
The DeviceIoControl function sends a control code directly to a specified device
driver, causing the corresponding device to perform the specified operation.

BOOL DeviceIoControl(
  HANDLE hDevice,             // handle to device of interest
  DWORD dwIoControlCode,      // control code of operation to perform
  LPVOID lpInBuffer,          // pointer to buffer to supply input data
  DWORD nInBufferSize,        // size, in bytes, of input buffer
  LPVOID lpOutBuffer,         // pointer to buffer to receive output data
  DWORD nOutBufferSize,       // size, in bytes, of output buffer
  LPDWORD lpBytesReturned,    // pointer to variable to receive byte count
  LPOVERLAPPED lpOverlapped   // pointer to structure for asynchronous operation
);
```

```
struct _ReadLenStruct{
    ULONG   DataLength;    // returns length of available data
    ULONG   FreeLength;    // returns the free space in to buffer
    ULONG   LFLength;      // returns the length of data available
                           // till the first Line feed
    ULONG   UserLength;    // returns the length of data available
                           // till the first user defined char.
    }    ReadLen;
unsigned long nBytes;
BOOLEAN success;
Unsigned char UserChar = ':';  // set user char to colon ':'


success = DeviceIoControl(hUsbDevice,
              IOCTL_LEPUSB_GET_READ_LENGTH,
              &UserChar, 1, &ReadLen, sizeof(ReadLen),
              &nBytes,
              NULL);
```

Set Timeout: - `IOCTL_LEPUSB_TIME_OUT`

One can set the driver read time out value by passing an unsigned long value to the time out set function. The units of this value are in milliseconds and the default is 30000 ms.

```
Ulong timeout = 30000;  // 30 seconds
success = DeviceIoControl(hUsbDevice,
              IOCTL_LEPUSB_TIME_OUT,
              &timeout, 1, &timeout, sizeof(timeout),
              &nBytes,
              NULL);
```

Get Driver Version: - `IOCTL_LEPUSB_GET_VERSION_INFO`

This function will allow an application to read the driver version. This functions returns "`LepUsb.sys Version: 1.00.2001.1, Feb 24 2000 16:41:02.`"

```
void PrintVersion(void )
      {
      UCHAR Ver[512];
      ULONG nBytes;
      BOOLEAN Success;

       Success = DeviceIoControl(hUsb, IOCTL_LEPUSB_GET_VERSION_INFO,
             NULL, 0, Ver, sizeof(Ver), &nBytes, NULL) ;
       if(!Success) printf("ERROR: Read Driver Version.\n");

       Ver[nBytes] = NULL;       // don't forget the NULL terminator
       printf("%s",Ver);         // printf the Version
      }
```

Signal Setup: - `IOCTL_LEPUSB_SIGNAL_SETUP`

This function will allow an application to be signaled when a certain event is satisfied. Events such as; data in receive buffer, line feed or user defined character found, or system not busy can be specified. This function will allow thread program to wait in efficient wait states till the event has occur. Thereby eliminating the constant polling for data. This command is preliminary, contact LEP for more information.

Registry Keys:

The following is a list of registry keys that may be edited. To access the registry run RegEdit.exe. The LEP USB registry keys can be located at:

```
HKEY_LOCAL_MACHINE\System\CurrentControlSet\Services\LepUsb\Parameters
```

LEP USB Registry Keys

- DebugLevel — Used for debugging, should always be zero.
- InBufferLen — The size in bytes of the receive buffer, default 512 bytes.
- TimeOut — The time out value in milliseconds, default 30000ms.
- DeviceName — This string contains device name of the last LEP USB device loaded. It can be used to open an interface to the LEP USB device.

LEP USB Programming Manual

Appendix A: Related Documents and Web sites:

Universal Serial Bus Specification → www.usb.org
Microsoft Development Network → www.msdn.microsoft.com
Surveying the New Win32 Driver Model for Windows 98 and Windows NT 5.0(Win2000). → www.msdn.microsoft.com/library/periodic/period97/html/wdm1211.htm
Driver Developemnt Kit Website → www.microsoft.com/ddk

LEP USB Programming Manual

Appendix B: Sample code in C:

This example runs in 32-bit console mode. It only scans for one device, then opens it, reads and displays the driver version. Then places the controller in high level mode, writes **ver** and **rconfig** to the LEP controller, wait for the answer, reads the answer, displays the returned string, and closes the handle.

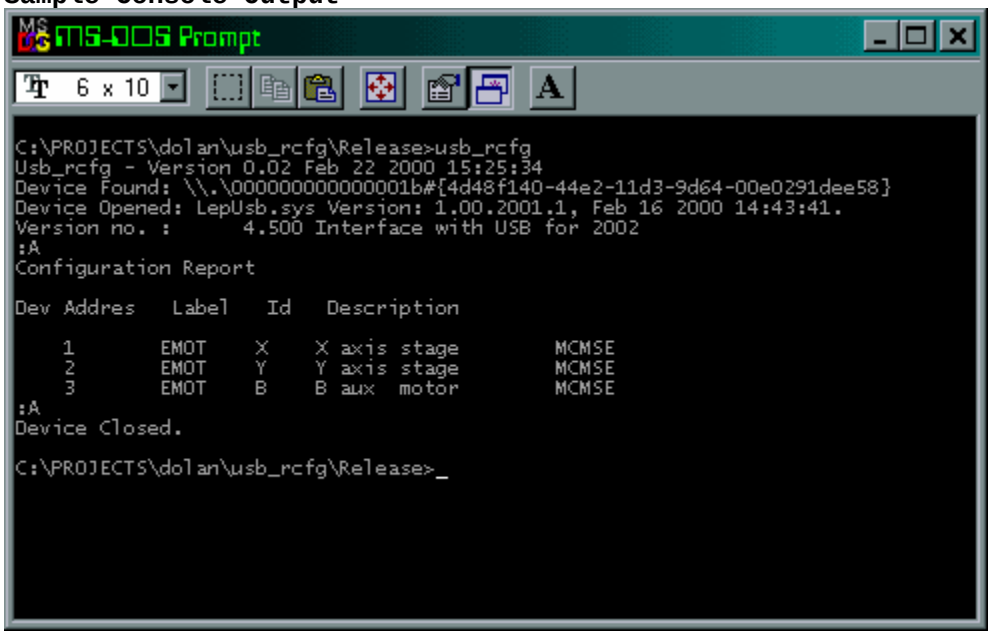
```
// usb_rcfg.cpp : A simple USB DOS console application.
// Author:                 Doug Lovett
// Birth:          02/07/2000
// Ludl Electronic Products Copyright 2000

#include "stdafx.h"
#include <objbase.h>
#include <setupapi.h>            // you may have to manually include this
library.
#include <initguid.h>
#include <conio.h>

// Defines
DEFINE_GUID(LEP_GUID,            // LEP's global, unique identifier
0x4d48f140, 0x44e2, 0x11d3, 0x9d, 0x64, 0x0, 0xe0, 0x29, 0x1d, 0xee, 0x58);

#define CTL_CODE( DeviceType, Function, Method, Access ) (                 \
   ((DeviceType) << 16) | ((Access) << 14) | ((Function) << 2) | (Method))

#define METHOD_BUFFERED                 0
#define FILE_ANY_ACCESS                 0
#define FILE_DEVICE_UNKNOWN             0x00000022


#define LEPUSB_IOCTL_VENDOR_INDEX       0x0800
#define IOCTL_LEPUSB_GET_READ_LENGTH    CTL_CODE(FILE_DEVICE_UNKNOWN,  \

LEPUSB_IOCTL_VENDOR_INDEX+1,\
                                                METHOD_BUFFERED,  \
                                                FILE_ANY_ACCESS)

#define IOCTL_LEPUSB_GET_VERSION_INFO    CTL_CODE(FILE_DEVICE_UNKNOWN,  \

LEPUSB_IOCTL_VENDOR_INDEX+2,\
                                                METHOD_BUFFERED,  \
                                                FILE_ANY_ACCESS)


// Function prototypes
int UsbScan(char* );
DWORD GetRxLen(char);
void PrintVersion(void);


// Global Variables
HANDLE hUsb;                     // Handle to the USB pipe

void main(int argc, char* argv[])
{
       char   DeviceName[MAX_PATH];
       char   Buffer[1024];
       DWORD  Length, result;
       printf("Usb_rcfg - Version 0.02 %s %s\n",__DATE__, __TIME__);

       // Scan for Usb Device
       if( !UsbScan(DeviceName) ) {
              printf("No devices found!\n");
              return;
              }

       printf("Device Found: %s\n",DeviceName);
```

```
        hUsb = CreateFile(DeviceName,
                     GENERIC_READ | GENERIC_WRITE,
                     FILE_SHARE_WRITE | FILE_SHARE_READ, NULL,
                              OPEN_EXISTING, 0, NULL);

       if( hUsb == INVALID_HANDLE_VALUE ) {
          printf("Error %d: Failed to open USB file handle.\n", GetLastError());
          return;
           }
        else
           printf("Device Opened: ");

        // Print driver version
        PrintVersion();

        // Put MAC2002 into High Level Mode
        sprintf(Buffer,"%c%c",0xFF,0x41);
        WriteFile(hUsb, Buffer, 2,&Length, NULL);


        // Send a 'ver' command
        sprintf(Buffer,"ver\n\r");
        if(!WriteFile(hUsb, Buffer,strlen(Buffer),&result, NULL))
               printf("ERROR writing to USB.\n");


        Length = GetRxLen(':');                 // wait for ':'
        if(Length )  {
               Length = GetRxLen(0);            // get total length
               ReadFile(hUsb, Buffer, Length, &result, NULL );
               Buffer[Length] = NULL; // add the NULL terminator
               printf("%s",Buffer);

               }


        // Send a 'rconfig' command
        sprintf(Buffer,"rconfig\n\r");
        if(!WriteFile(hUsb, Buffer,strlen(Buffer),&result, NULL))
               printf("ERROR writing to USB.\n");

        Length = GetRxLen(':');          // wait for ':'
        if(Length )  {
               Length = GetRxLen(0);     // get total length
               ReadFile(hUsb, Buffer, Length, &result, NULL );
               Buffer[Length] = NULL; // add NULL terminator
               printf("%s",Buffer );
               }


        CloseHandle(hUsb);
        printf("Device Closed.\n");
}       // end of main()


// ------------------------------------------------------------------------ //
// GetRxLen() - If the parameter passed is zero this function simply returns
// the available data in the USB IN buffer.  Else this function waits for the
// charactor given.  This work's well for high level commands where a colon(:)
// or linefeed (0x0A) is expected at the end of a incomming transmission.
// For low level mode you should just poll for the length of data that your
// command requires, because in low level command answers are fixed in length.
DWORD GetRxLen(char c )
        {
        struct _ReadLenStruct{
               union {
                      ULONG   DataLength;
                      char   UserChar[sizeof(ULONG)];
                      };
               ULONG   FreeLength;
               ULONG   LFLength;
               ULONG   UserLength;
```

```
            }           ReadLen;

      ULONG nBytes;
      BOOLEAN Success;

      while (1) {

            ReadLen.UserChar[0] = c;
            Success = DeviceIoControl(hUsb, IOCTL_LEPUSB_GET_READ_LENGTH,
             &ReadLen, 1, &ReadLen, sizeof(ReadLen),
             &nBytes, NULL) ;
            if(!Success || ( nBytes != sizeof(ReadLen) ) )
                  {
                  printf("ERROR: Returned from DeviceIoControl.\n");
                  CloseHandle(hUsb);
                  exit(0);
                  }

            if( ReadLen.UserLength || kbhit() ) break;
            if( !c ) break;
            }


      if( c )     return(ReadLen.UserLength);
      else  return(ReadLen.DataLength);

      }



// ------------------------------------------------------------------------ //
// UsbScan() -  Scan for first USB device with a GUID that matches the one
//              passed. Return 1 if device found, else 0.
//              This function only detech the first match of the device found.
//              If you have more than one controller attached you will have to
//              scan for the device more than once.  See the MFC windows
//              example. You can also get the devicename from the registry.
int UsbScan(char* DeviceName )
{
            // Get handle to the devices
      HDEVINFO  hInfo = SetupDiGetClassDevs((LPGUID)&LEP_GUID, NULL, NULL,
                                    DIGCF_PRESENT | DIGCF_INTERFACEDEVICE);

        if (hInfo == INVALID_HANDLE_VALUE)
            return(0);


        SP_INTERFACE_DEVICE_DATA Interface_Info;
        Interface_Info.cbSize = sizeof(Interface_Info);   // Enumerate device
        if (!SetupDiEnumInterfaceDevice(hInfo, NULL, (LPGUID) &LEP_GUID,0,
                                          &Interface_Info))
            {
            SetupDiDestroyDeviceInfoList(hInfo);
            return(0);
            }

        DWORD needed;                          // get the required length
        SetupDiGetInterfaceDeviceDetail(hInfo, &Interface_Info, NULL, 0,
                                    &needed, NULL);
        PSP_INTERFACE_DEVICE_DETAIL_DATA detail;
        detail = (PSP_INTERFACE_DEVICE_DETAIL_DATA) malloc(needed);
        if (!detail)
            {
            SetupDiDestroyDeviceInfoList(hInfo);
            return(0);
            }
                                          // fill the device details
        detail->cbSize = sizeof(SP_INTERFACE_DEVICE_DETAIL_DATA);
        if (!SetupDiGetInterfaceDeviceDetail(hInfo, &Interface_Info, detail,
                                       needed,NULL, NULL))
            {
            free((PVOID) detail);
```

```
            SetupDiDestroyDeviceInfoList(hInfo);
            return(0);
            }


         strncpy(DeviceName, detail->DevicePath, MAX_PATH );
         free((PVOID) detail);
         SetupDiDestroyDeviceInfoList(hInfo);
        return(1);                      // return true, USB device found

}


// ------------------------------------------------------------------------ //
// PrintVersion() - Gets and printf the driver version information.
void PrintVersion(void )
       {

       UCHAR Ver[512];
       ULONG nBytes;
       BOOLEAN Success;

      Success = DeviceIoControl(hUsb, IOCTL_LEPUSB_GET_VERSION_INFO,
              NULL, 0, Ver, sizeof(Ver), &nBytes, NULL) ;
      if(!Success) printf("ERROR: Read Driver Version.\n");

      Ver[nBytes] = NULL;       // don't forget the NULL terminator
      printf("%s",Ver);         // printf the Version

       }


// ------------------------------------------------------------------------ //
```

Sample Console Output



LEP USB Programming Manual

Appendix C: Sample code in Visual Basic:

This example runs in windows mode. It only scans for one device and opens it. Then the GUI allow the user to enter high level commands. When the program finds an enter key (CR) the program sends the command to the LEP USB device. On a timer event, the length of any available data in the IN buffer is checked. If data is present the data is displayed. This code is very basic and kept to a minimal to illustrate function.

MainForm Code

```
' LEP_USB_VB Example Code
' MainForm 0.001 02/23/2000
' Ludl Electronic Products Copyright 2000  - www.ludl.com
' This code makes function calls to the VB_BAS.bas file.


' Closes the device on exit
Private Sub ExitButton_Click()
Dim Status As Boolean
Status = CloseDevice()      ' close device
End
End Sub

' Initialize the program
Private Sub Form_Load()
Dim Status As Long
Status = OpenDevice()       ' Gets the device name and opens the device
DeviceNameLabel.Caption = VB_USB.lpDeviceName   ' Displays the device Name
Status = WriteDevice((Chr(255) & Chr(65)), 2)   ' Put Controller in High level
mode
If Status = 0 Then          ' Check for errors
    Status = CloseDevice()  ' if errors exit
    End
End If
Timer1.Interval = 200       ' Set timer to 200 mSecond interval
Timer1.Enabled = True       ' Enable the timer
End Sub

'Timer1 - Check for data in the IN buffer. If data present, get the data
'and display it in the RxText Box.
Private Sub Timer1_Timer()
On Error GoTo ERROR_HANDLER
Dim Length As Long
Dim ByteRead As Long
Dim RxData As String

Length = GetDeviceLength(0)         ' Get available data in the IN Buffer
If Length <> 0 Then                 ' If data exist
    RxData = Space(Length)          ' Allocate the space for the returned string
    BytesRead = ReadDevice(RxData, Length)  ' Get the IN data
    MainForm.RxText.Text = RxData   ' Copy it to the RxText Box
End If
Exit Sub                            ' Exit Sub
ERROR_HANDLER:
    MsgBox "RxButton ERROR #" & Str$(Err) & " : " & Error & Chr(13)
End Sub

'KeyPress - Checks to see if the keypressed was an CR.  If so, get the commmand
'and write it to the OUT buffer.
Private Sub TxText_KeyPress(Key As Integer)
On Error GoTo ERROR_HANDLER
Dim TxCommand As String
Dim TxLength As Long

If Key <> 13 Then
    Exit Sub                        ' check for CR, if none exit
End If
```

```
TxCommand = TxText.Text             ' get the command
TxCommand = TxCommand & Chr(13)     ' Add CR for MAC2002 high Level

TxLength = Len(TxCommand)           ' get the length
RxText.Text = TxCommand             ' display the command in the RxText Box

If TxLength <> WriteDevice(TxCommand, TxLength) Then    ' write the text
    MsgBox "TxButton ERROR length not equal"            ' check for error
End If
Exit Sub                            ' End Sub
ERROR_HANDLER:
    MsgBox "RxButton ERROR #" & Str$(Err) & " : " & Error & Chr(13)
End Sub
```

VB_USB Code

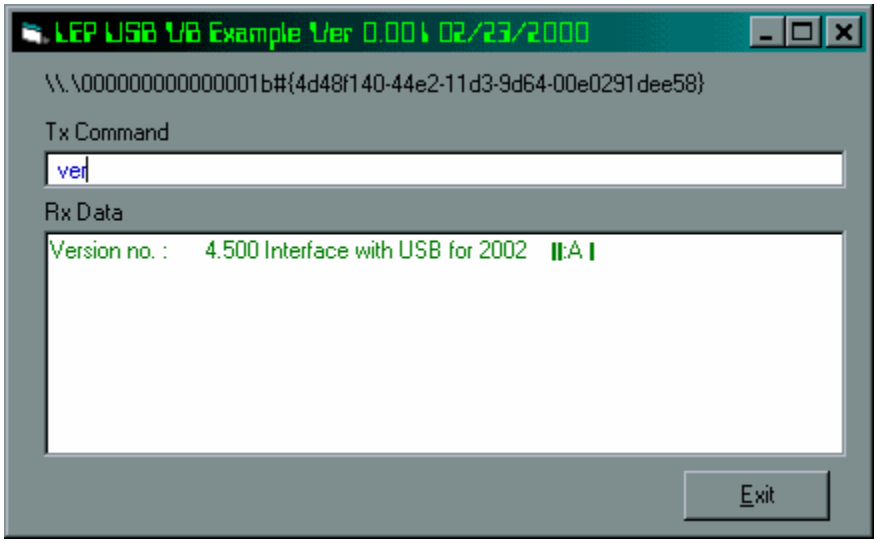
```
' Filename: VB_USB
' Author:   Doug Lovett
' Birth:    02/23/2000
' Ludl Electronic Products Copyright 2000  - www.ludl.com
' Subject:  Functions to access the LEP USB Device.
' Include this file in your VB project and call these functions
' to access the LEP USB device.

' Global data variables
Public hDevice As Long      'Handle to the device
Public lpDeviceName As String 'Copy of the device name

' Registry and File Constants
Const HKEY_LOCAL_MACHINE = &H80000002
Const GENERIC_READ = &H80000000
Const GENERIC_WRITE = &H40000000
Const FILE_SHARE_WRITE = &H2
Const FILE_SHARE_READ = &H1
Const OPEN_EXISTING = &H3
Const CTL_CODE_READ_LEN = &H222004

'Structure Needed for GetDeviceLength
Type GetLenStructure
    DataLength As Long
    FreeLength As Long
    LFLength   As Long
    UserLength As Long
End Type

' declare references to external procedures in a dynamic-link library (DLL).
Declare Function RegOpenKeyEx Lib "advapi32" Alias "RegOpenKeyExA" _
    (ByVal hKey As Long, ByVal lpSubKey As String, ByVal ulOptions As Long, _
    ByVal samDesired As Long, phkResult As Long) As Long

Declare Function RegQueryValueEx Lib "advapi32" Alias "RegQueryValueExA" _
    (ByVal hKey As Long, ByVal lpValueName As String, ByVal lpReserved As Long, _
    ByRef lpType As Long, ByVal szData As String, ByRef lpcbData As Long) As Long

Declare Function RegCloseKey Lib "advapi32" (ByVal hKey As Long) As Long

Declare Function CreateFile Lib "kernel32" Alias "CreateFileA" _
    (ByVal lpFileName As String, ByVal dwDesiredAccess As Long, _
     ByVal dwShareMode As Long, ByVal lpSecurityAttributes As Long, _
     ByVal dwCreationDisposition As Long, ByVal dwFlagsAndAttributes As Long, _
     ByVal hTemplateFile As Long) As Long

Declare Function CloseHandle Lib "kernel32" (ByVal hObject As Long) As Boolean

Declare Function DeviceIoControl Lib "kernel32" _
    (ByVal hDevice As Long, ByVal dwIocontrolCode As Long, _
     ByVal lpInBuffer As String, ByVal nInBufferSize As Long, _
     ByRef lpOutBuffer As GetLenStructure, ByVal nOutBufferSize As Long, _
     ByRef lpBytesReturned As Long, ByVal lpOverLapped As Long) As Boolean

Declare Function ReadFile Lib "kernel32" (ByVal hFile As Long, _
    ByVal lpBuffer As String, ByVal nNumberOfBytesToRead As Long, _
    ByRef lpNumberOfBytesRead As Long, ByVal lpOverLapped As Long) As Boolean

Declare Function WriteFile Lib "kernel32" (ByVal hFile As Long, _
    ByVal lpBuffer As String, ByVal nNumberOfBytesToWrite As Long, _
```

```
    ByRef lpNumberOfBytesWrtten As Long, ByVal lpOverLapped As Long) As Boolean

'OpenDevice - This function reads the device name from the registry and then
'opens the device and stores a handle to the device in hDevice. Returns zero
'on error.  This function also stores the full device name is DeviceName.
Function OpenDevice() As Boolean
On Error GoTo ERROR_HANDLER

'Get Device Name from the registry
lpDeviceName = GetRegValue(HKEY_LOCAL_MACHINE, _
                        "System\CurrentControlSet\Services\LepUsb\Parameters\", _
                        "DeviceName", "")
If lpDeviceName = "" Then   ' exit on error
    MsgBox "Unable to open device, check connection and power."
    lpDeviceName = "Device Not Found!"
    OpenDevice = 0
    Exit Function
End If
' Try and open the device. This will fail if device not present
hDevice = CreateFile(lpDeviceName, GENERIC_READ Or GENERIC_WRITE, _
                     FILE_SHARE_WRITE Or FILE_SHARE_READ, 0, _
                     OPEN_EXISTING, 0, 0)

If hDevice <= 0 Then    ' check for error
    MsgBox "Unable to open device, check connection and power"
    lpDeviceName = "Device Not Found!"
    OpenDevice = 0
End If

Exit Function
ERROR_HANDLER:
    MsgBox "OpenDevice() ERROR #" & Str$(Err) & " : " & Error
End Function

'CloseDevice - Closes the device, always close device after use.
'If you don't close the device after use, you will not be able
'to open it up again without cycle pluging the USB cable.
Function CloseDevice() As Boolean
On Error GoTo ERROR_HANDLER
CloseDevice = CloseHandle(hDevice)  ' Close the device
hDevice = 0                         ' Null the handle
If CloseDevice = False Then         ' Check for errors
    MsgBox "Error closing file"     ' Display errors
End If
Exit Function
ERROR_HANDLER:
    MsgBox "CloseDevice() ERROR #" & Str$(Err) & " : " & Error
End Function

' GetDeviceLength - This function returns the data in bytes available
' in the USB IN buffer, if the parameter passed is zero. Else it returns
' the data present up to and including the first match of the parameter
' passed.  If there is no match, zero is returned.  The later is well suited
' for high level commands were a linefeed of carign returns is expected
' at the end of an incoming tranmission.  For low level it would be better
' to poll the data length require, because in low level the data length is
' fixed.
Function GetDeviceLength(c As Byte) As Long
Dim lpDataBytes As Long
Dim nTestByte As Long
Dim TestByte As String
Dim GetLenObject As GetLenStructure

On Error GoTo ERROR_HANDLER
If hDevice <= 0 Then                ' check for valid handle
    MsgBox "GetDevicelength() Handle invalid!"
    Exit Function
End If

If c = 0 Then                       ' Setup the user search string
    TestByte = ""
    nTestByte = 0
Else
    TestByte = c
    nTestByte = 1
End If

                                    ' Call the read length function
If 0 = DeviceIoControl(hDevice, CTL_CODE_READ_LEN, TestByte, nTestByte, _
                       GetLenObject, Len(GetLenObject), lpDataBytes, 0) Then
    MsgBox "GetDeviceLength() Error calling DLL."
```

```
    GetDeviceLength = 0
    Exit Function
End If

If c = 0 Then                       ' Return the Length
    GetDeviceLength = GetLenObject.DataLength
Else
    GetDeviceLength = GetLenObject.UserLength
End If

Exit Function

ERROR_HANDLER:
    MsgBox "GetDeviceLength() ERROR #" & Str$(Err) & " : " & Error
End Function

'ReadDevice - Reads the number of bytes told, from the IN Usb buffer.
'If you ask for more bytes than are available in the IN buffer the
'function will wait till either the data comes or a timeout occurs.
'Returns the amount of data in the returned string.
'Timeout is defaulted to 30sec and may be changed with a DeviceIoControl
'function (see manual) or in the registry at
'(HKEY_LOCAL_MACHINE\System\CurrentControlSet\Services\LepUsb\Parameters\TimeOut).
'The default maximun size for the IN buffer is 512 bytes and can be
'changed with a DeviceIoControl (see manual) or in the registry, at
'(HKEY_LOCAL_MACHINE\System\CurrentControlSet\Services\LepUsb\Parameters\InBufferLen).
Function ReadDevice(lpInBuffer As String, lInLength As Long) As Long
Dim lpBytesRead As Long

On Error GoTo ERROR_HANDLER
If hDevice <= 0 Then                ' Check for valid handle
    MsgBox "ReadDevice() Handle invalid!"
    Exit Function
End If

If lInLength > Len(lpInBuffer) Then     ' Check string length
    MsgBox "GetDeviceLength() Error in size of string!"
    Exit Function
End If

                                    ' Read the Data
If 0 = ReadFile(hDevice, lpInBuffer, lInLength, lpBytesRead, 0) Then
    ReadDevice = 0
Else
    ReadDevice = lpBytesRead            ' Return the length read
End If

Exit Function
ERROR_HANDLER:
    MsgBox "ReadDevice() ERROR #" & Str$(Err) & " : " & Error

End Function

'WriteDevice - Writes the bytes amount told from the string passed to the
'USB OUT Buffer.  Returns the amount of bytes written.
Function WriteDevice(lpOutBuffer As String, lOutLength As Long) As Long
Dim lpBytesWritten As Long

On Error GoTo ERROR_HANDLER
If hDevice <= 0 Then                ' Check for valid handle
    MsgBox "WriteDevice() Handle invalid!"
    Exit Function
End If
                                    ' Write the Data to the device
If 0 = WriteFile(hDevice, lpOutBuffer, lOutLength, lpBytesWritten, 0) Then
    WriteDevice = 0
Else
    WriteDevice = lpBytesWritten        ' Return the data amount written
End If

Exit Function
ERROR_HANDLER:
    MsgBox "WriteDevice() ERROR #" & Str$(Err) & " : " & Error
End Function


' GetRegValue - Gets the Key value in the registry given a registry key.
Function GetRegValue(hKey As Long, lpszSubKey As String, szKey As String, _
                     szDefault As String) As String
On Error GoTo ERROR_HANDLER
```

```
Dim phkResult As Long, lResult As Long
Dim szBuffer As String, lBuffSize As Long

'Create Buffer
szBuffer = Space(255)           ' Allocate buffer space
lBuffSize = Len(szBuffer)       ' Set the length

                                'Open the Key
RegOpenKeyEx hKey, lpszSubKey, 0, 1, phkResult

                                'Query the value
lResult = RegQueryValueEx(phkResult, szKey, 0, 0, szBuffer, lBuffSize)

RegCloseKey phkResult           'Close the Key

'Return obtained value
If lResult = ERROR_SUCCESS Then
    GetRegValue = szBuffer
Else
    GetRegValue = szDefault
End If
Exit Function

ERROR_HANDLER:
    MsgBox "GetRegValue() ERROR #" & Str$(Err) & " : " & Error & Chr(13) _
    & "Please exit and try again."

    GetRegValue = szDefault
End Function
```

Sample Visual Basic Output



LEP USB Programming Manual

Appendix D: Sample code in C/C++ MFC Windows:

This example runs in windows mode. It can detect more than one device and mimics a terminal program.

Lep_USB.Cpp – Main window functions

```
// Filename:   UsbLep.Cpp
// Date:             02/23/00
// Author:           Doug Lovett
// Ludl Electronic Products Copyright 2000

// Include files
#include <afxwin.h>
#include <afxcmn.h>
#include <iostream.h>
#include <objbase.h>
#include <initguid.h>
#include "mfc.h"
#include "resource.h"



// The GUID for the LEP USB Device
// {00873FDF-61A8-11d1-AA5E-00C04FB1728B}  for LEPUSB.SYS
#define LEP_GUID_STR  "{4D48F140-44E2-11d3-9D64-00E0291DEE58}"



// --- Globals Variables ---
HANDLE hUsb = NULL;                         // handle to the main usb pipe
CMainEditWin editOb;                 // Global edit box object
CUsbCode CUsb;                              // Global Usb function class

// - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - //
// Initialize the Application
BOOL CApp::InitInstance()
{

               LPCTSTR cname = AfxRegisterWndClass( 0, NULL, NULL,
                                    LoadIcon(MAKEINTRESOURCE(IDI_ICON1)));


        m_pMainWnd = new CMainWin(cname);
            m_pMainWnd->ShowWindow(m_nCmdShow);
        m_pMainWnd->UpdateWindow();


               // scan for the USD devices
               COpenDeviceDialog diagOb( MAKEINTRESOURCE(IDD_DIALOG1), m_pMainWnd );
               diagOb.DoModal();             // activate modal dialog box
               editOb.SetFocus( );

        return TRUE;
}
CApp App;                                              //Instantiate the application

// - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - //
// class CMainWin : public CFrameWnd
// - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - //
// The Application's Main Message Map
BEGIN_MESSAGE_MAP( CMainWin, CFrameWnd )
        ON_WM_PAINT()
        ON_WM_SIZE()
        ON_WM_ACTIVATE()
        ON_WM_TIMER()
        ON_COMMAND(IDM_DEVICE_OPEN, OnDeviceOpen)
        ON_COMMAND(IDM_DEVICE_CLOSE, OnDeviceClose)
        ON_COMMAND(IDM_EXIT, OnExit)
        ON_COMMAND(IDM_DEVICE_CLEAR, OnClear)
        ON_COMMAND(IDM_TEST, OnTest)
END_MESSAGE_MAP()

// CMainWin Consturct - Setups up the main window
CMainWin::CMainWin(LPCSTR ClassName)
{
               CString CTitle;
               CTitle = "USB Terminal Program 0.04, ";
               CTitle += (CString)__DATE__ +" "+(CString)__TIME__;
```

```
        Create(ClassName, CTitle, WS_OVERLAPPEDWINDOW,
                rectDefault, NULL, MAKEINTRESOURCE(IDR_MENU1) );

                hUsb = INVALID_HANDLE_VALUE;  // Null are USB handle

                RECT EditBox;                                   // create an EditBox
structure
                EditBox.left = 0;                               // set box size
                EditBox.top = 0;
                EditBox.right  = GetSystemMetrics(SM_CXSCREEN);
                EditBox.bottom = GetSystemMetrics(SM_CYSCREEN);

                // create an edit object
      editOb.Create(WS_VSCROLL | WS_HSCROLL | ES_AUTOHSCROLL | ES_AUTOVSCROLL|
                ES_LEFT | ES_MULTILINE | ES_NOHIDESEL | ES_WANTRETURN,
                                        EditBox, this, IDC_EDIT_WIN );
                editOb.ShowWindow(SW_SHOW);
                editOb.SetFocus( );


                CHARFORMAT cf;
                editOb.GetDefaultCharFormat( cf );
                cf.dwMask = CFM_COLOR;
                cf.dwEffects = cf.dwEffects & ~CFE_AUTOCOLOR;
                cf.crTextColor = editOb.TextColor = RGB(255,0,0);    //RED
                editOb.SetWordCharFormat( cf );
                editOb.SetDefaultCharFormat(cf);
                CTitle += "\n";
                editOb.ReplaceSel( CTitle );         // Print the Title



}

// OnTimer Event - On event this function checks for data in the IN USB
// buffer.  If data exist, data is read from the device buffer and print
// to the Main Edit Object.
afx_msg void CMainWin::OnTimer( UINT ID )
{

        DWORD length = CUsb.GetLen();                // Get the available data length
        if( length )                                                // If data? get it
                {
                char *pData = new char[length+1];    // Allocate the buffer
                length = CUsb.Read(pData, length);   // Get the data
                pData[length] = NULL;                        // Don't forget to Null
terminate it!
                editOb.GreenTextOut(pData);                  // print it out in green text
                delete(pData);                                  // deallocate the mem
                }

}

// On Main window activation set focus in the edit object
afx_msg void CMainWin::OnActivate(UINT nState, CWnd* pWndOther, BOOL bMinimized)
{

        CWnd::OnActivate( nState, pWndOther, bMinimized );
        if( (nState == WA_ACTIVE) || (nState == WA_CLICKACTIVE) )
                {
                editOb.SetFocus( );           // always place the focus in the edit box
                }
}

// Clear Event - Clears the Edit Objects contents
afx_msg void CMainWin::OnClear()
{

        editOb.SetWindowText("");

}

// OnTest Event - used for debugging to turn off thr timer - Currently unreachable!
afx_msg void CMainWin::OnTest(void)
{
        if( IDYES == MessageBox("Timer?","Timer",MB_ICONQUESTION |MB_YESNOCANCEL) )
                App.m_pMainWnd->SetTimer(ID_Timer,30,NULL);  //30ms
        else
                App.m_pMainWnd->KillTimer(ID_Timer);
}
```

```
// On Close Event - close device
afx_msg void CMainWin::OnDeviceClose()
{
    CUsb.Close();
}


// Open Device Event - Opens the device and set focus to the main edit object
afx_msg void CMainWin::OnDeviceOpen()
{
   COpenDeviceDialog diagOb( MAKEINTRESOURCE(IDD_DIALOG1), this );
        diagOb.DoModal();      // activate modal dialog box
        editOb.SetFocus( );
}


// Exit event - Send the WM_CLOSE msg that closes the program
afx_msg void CMainWin::OnExit()
{
        int response = IDYES;
        //response = MessageBox("Quit?","Exit", MB_YESNO );
        if( response == IDYES )
                SendMessage(WM_CLOSE); // terminate program
}

// OnPaint event - Update the screen using contents of virtual window
afx_msg void CMainWin::OnPaint()
{
        CFrameWnd::OnPaint();
}


// On Size event - on size reset the window pos
afx_msg void CMainWin::OnSize( UINT nType, int cx, int cy )
{
        editOb.SetWindowPos( NULL, 0, 0, cx, cy, NULL );
}



// - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - //
// class CMainEditWin : public CEdit
// - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - //
// Main edit box object
BEGIN_MESSAGE_MAP( CMainEditWin, CRichEditCtrl )
        ON_WM_CHAR()
END_MESSAGE_MAP()

CMainEditWin::CMainEditWin()
{
                // make the edit box
                CRichEditCtrl();       // constructor
}

// Onchar event
void CMainEditWin::OnChar(UINT ch, UINT count, UINT flags)
{
        int response = IDYES;
        long nLine;
        char str[1024];

        // Check for CR, if found, send line to USB device
        if(ch == 0x0D)
                {
                nLine =  LineFromChar( -1 );                                    //
select current line
                response = GetLine( nLine, str, sizeof(str)-2);             // get the
line
                if((str[response-3] == str[response-2]) && (str[response-2] == 0x0D) )
                        {
                        str[response-2] = 0x0A;                       // strips off the
double LF
                        str[response-1] = NULL;                       // Null terminate
                        }
                else
                        str[response] = NULL;                 // Null terminate
                CUsb.Write(str);                                      // Send the line to
the USB device
```

```
                }

        editOb.BlueTextOut("");                                         // set blue color
        CRichEditCtrl::OnChar(ch,count,flags);       // send char down to the lower class

        // currently edit object is not limited in size, should limit it here
        // if (getnumberlines() > MAXLINES) delete oldest line.

}


// OnCacel event
void CMainEditWin::OnCancel()
{
        KillTimer(ID_Timer);
        CUsb.Close();
        DestroyWindow();
}


// print text to the edit object in red
void CMainEditWin::RedTextOut(char * text)
        {
        CHARFORMAT cf;
        GetSelectionCharFormat( cf );
        // check color, change if need be
        TextColor = RGB(255,0,0);     // Red
        if( cf.crTextColor != editOb.TextColor )
                {
                cf.dwMask = CFM_COLOR;
                cf.crTextColor = TextColor;
                SetWordCharFormat( cf );
                }
        editOb.ReplaceSel( text );
        }

// print text to the edit object in blue
void CMainEditWin::BlueTextOut(char * text)
        {
        CHARFORMAT cf;
        GetSelectionCharFormat( cf );
        // check color, change if need be
        TextColor = RGB(0,0,255);     // Blue
        if( cf.crTextColor != editOb.TextColor )
                {
                cf.dwMask = CFM_COLOR;
                cf.crTextColor = TextColor;
                SetWordCharFormat( cf );
                }
        editOb.ReplaceSel( text );
        }

// print text to the edit object in green
void CMainEditWin::GreenTextOut(char * text)
        {
        CHARFORMAT cf;
        editOb.GetSelectionCharFormat( cf );
        // check color, change if need be
        editOb.TextColor = RGB(0,100,0);
        if( cf.crTextColor != editOb.TextColor )
                {
                cf.dwMask = CFM_COLOR;
                cf.crTextColor = editOb.TextColor;
                editOb.SetWordCharFormat( cf );
                }
        editOb.ReplaceSel( text );
        }

// - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - //
// class COpenDeviceDialog : public CDialog
// - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - //
// Popup window to select USB device

// Open Device Dialog message map
BEGIN_MESSAGE_MAP( COpenDeviceDialog, CDialog)
        ON_COMMAND(ID_OPEN_GUID, OnSelect )
        ON_LBN_DBLCLK( IDC_LIST1, OnSelect)
        ON_COMMAND(IDC_RESCAN, OnReScan )
END_MESSAGE_MAP()
```

```
// Init Dialog box
BOOL COpenDeviceDialog::OnInitDialog()
{
	int i,j=0;
	CString Devices[10];
	CDialog::OnInitDialog();		// call the base class initialize first

	// initialize the edit box
	CEdit* ebptr = (CEdit*)GetDlgItem(IDC_EDIT1);	// link the edit box
	Devices[0] = LEP_GUID_STR;					// get the
device LEP GUID
	ebptr->SetWindowText(Devices[0]);				// display the GUID

	i = CUsb.Scan( Devices );					// scan for
the devices
	if(!i)
	// if none found, exit
		{
		MessageBox("Unable to enumerate Usb device","Enumeration Error");
		SendMessage(WM_CLOSE);
		return(0);
		}
	CListBox* lbptr = (CListBox *) GetDlgItem(IDC_LIST1);	// link the list box
	// init the list box
	while(j<i)
	// fill the list box with
		lbptr->AddString(Devices[j++]);				// the device
names

	lbptr->SetCurSel(0);						// default to
the first
	return TRUE;
}


// Open device dialog call backs
// OnSelect button
afx_msg void COpenDeviceDialog::OnSelect()
{
	CString cmsg;
	CListBox* lbptr = (CListBox*)GetDlgItem(IDC_LIST1);
	int i;
	i = lbptr->GetCurSel();		// get the index of the selected device name
	if(i == LB_ERR)	{
		SendMessage(WM_CLOSE);// if error, exit
		return;
		}
	else{
		lbptr->GetText(i,cmsg);	// get the device name
		CUsb.Open( cmsg );		// open the device
		}
	SendMessage(WM_CLOSE);	// close the dialog box
}



// On Rescan event button - rescan for the device names
afx_msg void COpenDeviceDialog::OnReScan()
{
	int i,j=0;
	CString Devices[10];
	CEdit* ebptr = (CEdit*)GetDlgItem(IDC_EDIT1);
	CListBox* lbptr = (CListBox *) GetDlgItem(IDC_LIST1);

	// Get the GUID
	ebptr->GetWindowText(Devices[0]);

	while( lbptr->DeleteString(0))	;		// delete all the devices
	i = CUsb.Scan( Devices );
	if(!i)
		{
		MessageBox("Unable to enumerate Usb device","Enumeration Error");
		SendMessage(WM_CLOSE);
		}

	// init the list box
	while(j<i)	// fill the boxes
		lbptr->AddString(Devices[j++]);
	lbptr->SetCurSel(0);
}
```

```
// - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - //
```

UsbCode.Cpp – USB functions

```
//     Filename:      usbcode.cpp
//     Birth:         7/16/99
//     Author:        Doug Lovett
//     Ludl Electronic Products Copyright 2000

// Includes header
#include <afxwin.h>
#include <afxcmn.h>
#include <iostream.h>
#include <afxtempl.h>
#include <objbase.h>
#include <setupapi.h>
#include "devioctl.h"

#include <initguid.h>
#include "resource.h"
#include "mfc.h"

// CTL_CODE defines
#define LEPUSB_IOCTL_INDEX         0x0000
#define LEPUSB_IOCTL_VENDOR_INDEX  0x0800


#define IOCTL_LEPUSB_GET_CONFIG_DESCRIPTOR     CTL_CODE(FILE_DEVICE_UNKNOWN, \
                                                    LEPUSB_IOCTL_INDEX,\
                                                    METHOD_BUFFERED,  \
                                                    FILE_ANY_ACCESS)

#define IOCTL_LEPUSB_RESET_DEVICE       CTL_CODE(FILE_DEVICE_UNKNOWN,  \
                                                    LEPUSB_IOCTL_INDEX+1,\
                                                    METHOD_BUFFERED,  \
                                                    FILE_ANY_ACCESS)

#define IOCTL_LEPUSB_RESET_PIPE         CTL_CODE(FILE_DEVICE_UNKNOWN,  \
                                                    LEPUSB_IOCTL_INDEX+2,\
                                                    METHOD_BUFFERED,  \
                                                    FILE_ANY_ACCESS)


#define IOCTL_LEPUSB_TIME_OUT           CTL_CODE(FILE_DEVICE_UNKNOWN,  \
                                                    LEPUSB_IOCTL_VENDOR_INDEX,\
                                                    METHOD_BUFFERED,  \
                                                    FILE_ANY_ACCESS)

#define IOCTL_LEPUSB_GET_READ_LENGTH    CTL_CODE(FILE_DEVICE_UNKNOWN,  \

LEPUSB_IOCTL_VENDOR_INDEX+1,\
                                                    METHOD_BUFFERED,  \
                                                    FILE_ANY_ACCESS)

#define IOCTL_LEPUSB_GET_VERSION_INFO    CTL_CODE(FILE_DEVICE_UNKNOWN,  \

LEPUSB_IOCTL_VENDOR_INDEX+2,\
                                                    METHOD_BUFFERED,  \
                                                    FILE_ANY_ACCESS)


// LEP USB Device GUID => {4D48F140-44E2-11d3-9D64-00E0291DEE58}
DEFINE_GUID(LEP_GUID,
0x4d48f140, 0x44e2, 0x11d3, 0x9d, 0x64, 0x0, 0xe0, 0x29, 0x1d, 0xee, 0x58);



// external referenced objects
extern CMainEditWin editOb;
extern HANDLE hUsb;
extern CApp App;
```

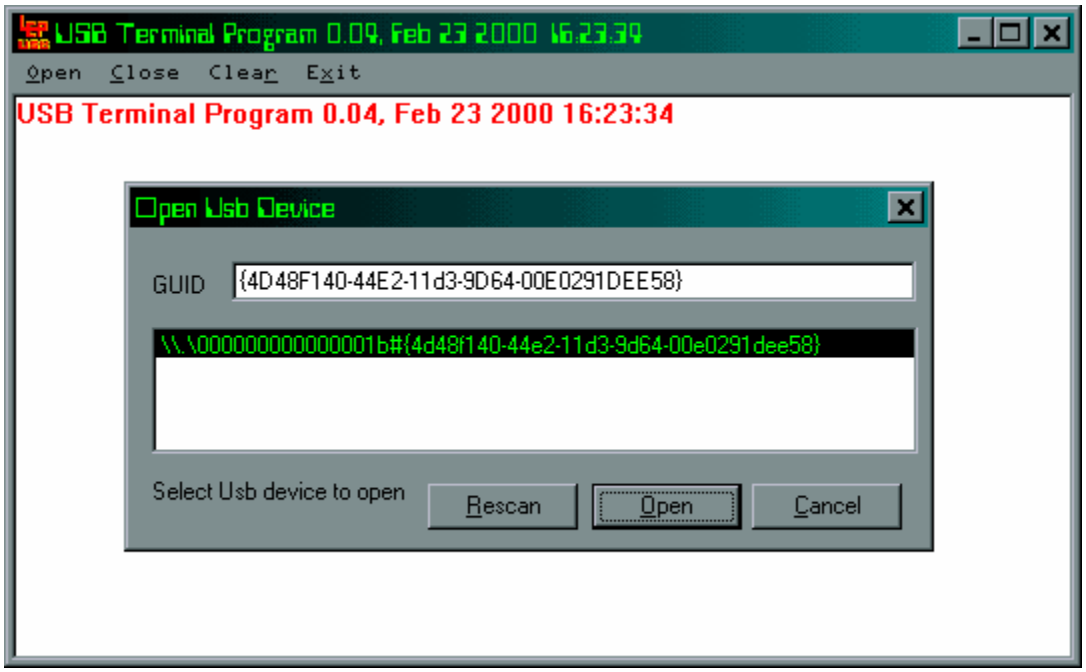
```
// CUsbCode Object Functions

CUsbCode::CUsbCode()          // Constructor
{
       FirstTimeOpened = 1;   // flag for first time
}


// Scans for all the device that match the GUID passed.
// returns an array of cStrings, containing the device names found,
// and returns the number of device names found. Returns zero if no
// devices found.
int     CUsbCode::Scan(CString * Devices)
{
       // Get handle to the devices manager
       HDEVINFO  hInfo = SetupDiGetClassDevs((LPGUID)&LEP_GUID, NULL, NULL,
                                             DIGCF_PRESENT |  DIGCF_INTERFACEDEVICE);
    if (hInfo == INVALID_HANDLE_VALUE)              // check for error
              return(0);

       for (DWORD i=0; ; ++i)                              // loop till all devices are
found
              {
              SP_INTERFACE_DEVICE_DATA Interface_Info;                    // declare
structure
              Interface_Info.cbSize = sizeof(Interface_Info);             // Enumerate
device
              if (!SetupDiEnumInterfaceDevice(hInfo, NULL, (LPGUID) &LEP_GUID,i,
&Interface_Info))
                     {
                     SetupDiDestroyDeviceInfoList(hInfo);
                     return(i);
                     }

              DWORD needed;
       // get the required lenght
              SetupDiGetInterfaceDeviceDetail(hInfo, &Interface_Info, NULL, 0, &needed,
NULL);
              PSP_INTERFACE_DEVICE_DETAIL_DATA detail =
(PSP_INTERFACE_DEVICE_DETAIL_DATA) malloc(needed);
              if (!detail)
                     {
                     SetupDiDestroyDeviceInfoList(hInfo);
                     return(i);
                     }

                     // fill the device details
              detail->cbSize = sizeof(SP_INTERFACE_DEVICE_DETAIL_DATA);
              if (!SetupDiGetInterfaceDeviceDetail(hInfo, &Interface_Info, detail,
needed,NULL, NULL))
                     {
                     free((PVOID) detail);
                     SetupDiDestroyDeviceInfoList(hInfo);
                     return(i);
                     }

              char name[MAX_PATH];
              strncpy(name, detail->DevicePath, sizeof(name));    // copy the device
name
              free((PVOID) detail);
       // free the mem
              Devices[i] = name;
       // save device name

       }  // end of for loop
}

// Open the device with the given device name
void CUsbCode::Open(CString DeviceName)
{
       // Create the handle to our device
       hUsb = CreateFile(DeviceName,
                     GENERIC_READ | GENERIC_WRITE,
                     FILE_SHARE_WRITE | FILE_SHARE_READ, NULL,
                            OPEN_EXISTING, 0, NULL);

    if( hUsb == INVALID_HANDLE_VALUE )       // process error
              {
              LPVOID lpMsgBuf;
              FormatMessage(
```

```
                        FORMAT_MESSAGE_ALLOCATE_BUFFER | FORMAT_MESSAGE_FROM_SYSTEM,
                        NULL, GetLastError(),
                        MAKELANGID(LANG_NEUTRAL, SUBLANG_DEFAULT), // Default language
                    (LPTSTR) &lpMsgBuf, 0, NULL );
                // Display the string.
                MessageBox(NULL,(LPTSTR)lpMsgBuf,"Failed to open Device",MB_ICONHAND);
                LocalFree( lpMsgBuf );  // Free the buffer.
                editOb.SetFocus( );
                }
        else    {
                if(FirstTimeOpened){                 // if this is the first time print
the driver version
                        ULONG nBytes;
                        if( !DeviceIoControl(hUsb, IOCTL_LEPUSB_GET_VERSION_INFO,
                              NULL, 0, msg, sizeof(msg), &nBytes, NULL))
                                editOb.RedTextOut( "ERROR: Reading Driver Version.\n");
                        msg[nBytes] = NULL;                          // don't forget the
NULL terminator
                        editOb.RedTextOut( msg);             // print the driver info
                        FirstTimeOpened = 0;
                        }
                editOb.RedTextOut( "Device opened.\n");
                sprintf(msg,"%c%c",0xFF,0x41);
                Write(msg);                                                  // switch to
high level (ascii) mode
                CMenu* pCMenu = App.m_pMainWnd->GetMenu();
                pCMenu->EnableMenuItem( IDM_DEVICE_CLOSE,MF_BYCOMMAND | MF_ENABLED);
                pCMenu->EnableMenuItem( IDM_DEVICE_OPEN , MF_BYCOMMAND | MF_GRAYED);
                if(!App.m_pMainWnd->SetTimer(ID_Timer,30,NULL))     //enable the timer for
30ms
                        editOb.RedTextOut("Unable to start read length timer!\n");
                }

        editOb.SetFocus( );

}


// Close the device,
int CUsbCode::Close()
{

        App.m_pMainWnd->KillTimer(ID_Timer);         // kills the timer
        if( CloseHandle( hUsb) )                                 // close the handle to
our device
                {
                editOb.RedTextOut( "Device closed.\n");
                CMenu* pCMenu = App.m_pMainWnd->GetMenu();
                pCMenu->EnableMenuItem( IDM_DEVICE_OPEN,  MF_BYCOMMAND | MF_ENABLED);
                pCMenu->EnableMenuItem( IDM_DEVICE_CLOSE, MF_BYCOMMAND | MF_GRAYED);
                App.m_pMainWnd->DrawMenuBar();
                editOb.SetFocus( );
                return(1);
                }
        else
                {
                LPVOID lpMsgBuf;
                FormatMessage(
                        FORMAT_MESSAGE_ALLOCATE_BUFFER | FORMAT_MESSAGE_FROM_SYSTEM,
                        NULL, GetLastError(),
                        MAKELANGID(LANG_NEUTRAL, SUBLANG_DEFAULT), // Default language
                    (LPTSTR) &lpMsgBuf, 0, NULL );
                // Display the string.
                MessageBox(NULL,(LPTSTR)lpMsgBuf,"Failed to close Device",MB_ICONHAND);
                LocalFree( lpMsgBuf );  // Free the buffer.
                editOb.SetFocus( );
                }

        return 0;
}

// Write's a string to the usb out port, string must be NULL terminated
// returns number of bytes written
DWORD CUsbCode::Write(char *pText)
{
        DWORD nBytesWritten;

        if(hUsb == INVALID_HANDLE_VALUE)             // check for valid handle
                {
                MessageBox(NULL,"USB OUT Device closed.","Failed to write to
Device",MB_ICONHAND);
```

```
                editOb.SetFocus( );
                return(0);
                }

        if(WriteFile(hUsb, pText, strlen(pText), &nBytesWritten, NULL))           //
write the string
                return nBytesWritten;
        else
                {               // process errors
                LPVOID lpMsgBuf;
                FormatMessage(
                        FORMAT_MESSAGE_ALLOCATE_BUFFER | FORMAT_MESSAGE_FROM_SYSTEM,
                        NULL, GetLastError(),
                        MAKELANGID(LANG_NEUTRAL, SUBLANG_DEFAULT), // Default language
                    (LPTSTR) &lpMsgBuf, 0, NULL );
                // Display the string.
                MessageBox(NULL,(LPTSTR)lpMsgBuf,"Failed to write Device",MB_ICONHAND);
                LocalFree( lpMsgBuf );  // Free the buffer.
                editOb.SetFocus( );
                return(nBytesWritten);
                }
}


// Reads data from the USB IN buffer into the memory pointer passed.
// Memory MUST be allocated for at least the size of the request.
// Returns the amount of data read from the IN buffer.
// If there is not enough data in the IN buffer, the readfile() function
// will wait till either; the data becomes available, or the function times out.
// Default timeout is 30 seconds, and can be adjust with the registry or
// a function call.
DWORD CUsbCode::Read(char * pText, DWORD nBytes)
{
        DWORD nBytesRead;

        if(hUsb == INVALID_HANDLE_VALUE)
                {
                App.m_pMainWnd->KillTimer(ID_Timer);
                MessageBox(NULL,"USB In Device closed.","Failed to read to
Device",MB_ICONHAND);
                editOb.SetFocus( );
                return(0);
                }

        if(ReadFile(hUsb, pText, nBytes, &nBytesRead, NULL))
                return nBytesRead;
        else
                {
                LPVOID lpMsgBuf;
                FormatMessage(
                        FORMAT_MESSAGE_ALLOCATE_BUFFER | FORMAT_MESSAGE_FROM_SYSTEM,
                        NULL, GetLastError(),
                        MAKELANGID(LANG_NEUTRAL, SUBLANG_DEFAULT), // Default language
                    (LPTSTR) &lpMsgBuf, 0, NULL );
                // Display the string.
                MessageBox(NULL,(LPTSTR)lpMsgBuf,"Failed to read Device",MB_ICONHAND);
                LocalFree( lpMsgBuf );  // Free the buffer.
                editOb.SetFocus( );
                return(nBytesRead);
                }
        return 0;
}

// GetLen - This function returns the length of data available in the
// USB IN buffer.
DWORD CUsbCode::GetLen()
{
        struct _ReadLenStruct{
    ULONG   DataLength;
    ULONG   FreeLength;
    ULONG   LFLength;
    ULONG   UserLength;
    }           ReadLen;
        unsigned long nBytes;
        BOOLEAN success;

    success = DeviceIoControl(hUsb,                 // call the get length function
              IOCTL_LEPUSB_GET_READ_LENGTH,
              NULL, 0, &ReadLen, sizeof(ReadLen),
              &nBytes,
              NULL);
```

```
        if( nBytes != sizeof(ReadLen) || !success )  // check of errors
                {
                App.m_pMainWnd->KillTimer(ID_Timer);
                LPVOID lpMsgBuf;
                FormatMessage(
                        FORMAT_MESSAGE_ALLOCATE_BUFFER | FORMAT_MESSAGE_FROM_SYSTEM,
                        NULL, GetLastError(),
                        MAKELANGID(LANG_NEUTRAL, SUBLANG_DEFAULT), // Default language
                    (LPTSTR) &lpMsgBuf, 0, NULL );
                // Display the string.
                MessageBox(NULL,(LPTSTR)lpMsgBuf,"Failed to read device
length!",MB_ICONHAND);
                LocalFree( lpMsgBuf );  // Free the buffer.
                editOb.SetFocus( );
                editOb.RedTextOut("Error reading receive length!\n");
                Close();
                return(0);
                }
        else
                return(ReadLen.DataLength);          // return the available data length
}

// ------------------------------------------------------------- //
```

MFC C/C++ Screen Output example



LEP USB Programming Manual

Appendix E: USB Bandwidth on the MAC2002 USB Controller

The MAC2002 USB controller uses the bulk transfer type USB communication protocol. The communication bandwidth of the MAC2002 USB controller depends on two variables; the first is the data packet size and the second is how much space is available in the USB frame. The latter says that the more devices you have on a USB port, the more the USB frame is used up and the less bandwidth we have. This is obviously an unknown factor, so to simplify things we will assume that the MAC2002 USB Controller is the only device on the USB bus. Just keep in mind that the more USB devices on the bus the less bandwidth we have.

The other factor is the data packet size. The USB protocol sends packet of data out on each USB frame, therefore the larger the data packet the higher the bandwidth. The average data packet size for the MAC2002 USB controller for high level mode is 20 bytes and for low level mode is 6 bytes. So for high level we can expect and bandwidth of 816 Kbytes or better and low level a bandwidth of 352 Kbytes or better. The table below list bandwidth at different data packet sizes.

USB Bulk Transfer Bandwidth

Data Packet Size	Bytes/Second
1	107000
2	200000
4	352000
8	568000
16	816000
32	105000
64	1210000

Note these values are theoretical and only represent the bandwidth of the USB communication protocol in one way. There are other factors that will decrease these values, such as; host computer speed and application load, MAC2002 Controller Interface loading and communication delays, and communication errors (such as in noisy environments).