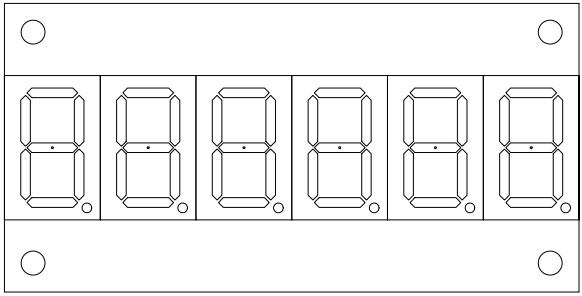
USB Numeric Display
Development Manual



Delcom Engineering
200 William St. Suite 302
Port Chester, NY
10573
914-934-5170
914-934-5171 Fax
www.delcom-eng.com

Document Ver USBNDSPY 1.1 June 31, 2003

Contents

1.0 Introduction

The USB Numeric Display visual and optional auditory signal device powered and activated via the USB port. The device has 4 to 8 seven segment LED digits. LED color is available in red and green. LED scan rate, flash rate and duty cycle are programmable. The optional auditory signal is produced by an piezo buzzer. Buzzer frequency, duty cycle and repeat count are all programmable. The device is self-power from the USB bus and requires no additional hardware.

2.0 Quick Start Up Guide

To start using the USB Numeric Display, simply plug it in to an available USB port on your computer. All USB devices are "hot plugable" so you can plug it in with the computer on or off. Windows™ will automatically detect the new device and search for the required driver. If the driver is not already installed on your computer Windows™ will display "New hardware Found" dialog box and prompts you to enter the location of the USB driver. Place the USB Numeric Display distribution disk in the floppy drive and select that drive to install the driver. This procedure for loading the driver is only required once. Once the driver is loaded you can use one of the sample applications on the distribution disk to control the USB Numeric Display device.
If you have problems installing the driver please refer to "Loading Windows™ USB Drivers" applications note on the web site.

For the latest driver and updates refer to the Delcom web site.

3.0 Specifications

3.1 OS Compatibility

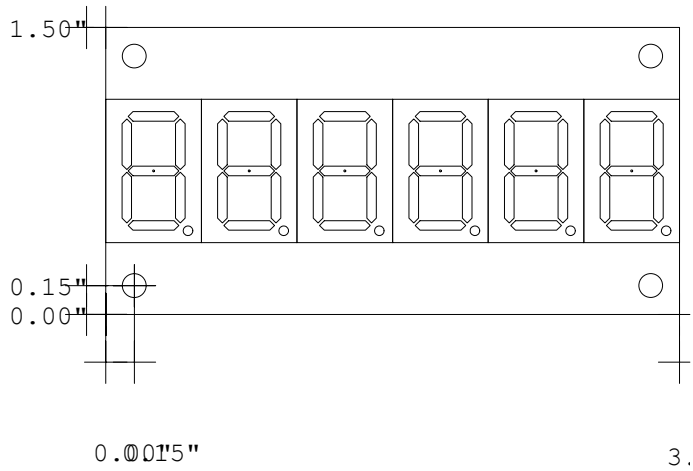
The USB Numeric Display device is compatible with the following operating systems. Windows 98, ME, 2000, XP and MAC OSX.

3.2 Mounting

The USB Numeric Display can be purchased with or with out an enclosure. The dimension for the PCB and enclosure are shown below.

3.2 Mechanical

The USB Numeric Display dimensions, PCB only.



The USB Numeric Display dimensions with enclosure are 3.9” width, 2.0” height and 0.8” deep.

3.3 Electrical

The USB Visual Indicator consumes 8mA when all LEDs and buzzer is off and 200ma maximum when all LEDs are on. The USB cable length is 2 meters (6 ½ Feet). {Also see 9.1}

3.4 Visual Output

The LEDs have a typical rating of 100,000 hours.

3.5 Audio Output

The audio signal is generated from a piezo buzzer producing 85 dB @ 2.4KHz with out the enclosure.

4.0 Features

4.1 Numeric Display

The USB Numeric Display can be programmed in three modes. There are Raw, Hex and Ascii. In Raw data is sent directly to the seven segment Leds with out any conversion. See the Raw data table below. Raw mode allows the programmer to have complete control over each Led segment. This allow's the to create custom characters. Valid values for Raw mode are 0-255 where 0 is all LEDs off.
Hex mode allows the user to enter a number from 0 to 15. Values of 10 through 15 are displayed as A,bC,d,E. Valid range is 0-15, any other value will show as a hyphen.
Ascii mode allow the user to send Ascii character to the device. Valid Ascii characters are 0-9, A-F and a-f. Any other character will be displayed as a hyphen.

When send data to the device the data length must be set to the correct number of digit that you what to display. All other digits greater than the length are turned off.

The Led scan rate may be adjusted through software. The default is 10 and the units are in 128us. Valid range is 1-255, where 255 is the slowest scan rate.

The display can be programmer to flash. Valid flash values are from 0 to 255, when zero equals flash off and 255 is the slowest flash rate. Flash in off by default.

Each device can be configured as a 1 to 8 digit display. The default display value is 6. The display setup command can change the device from a 1 to 8 digit display. When less than 8 digits are used the unused scan lines can be used as general I/O.

The decimal point has an separate command to control where the decimal point goes. Decimal point can also be controller direct in Raw mode.

Table1 – Bit to Segment Relationship

8	7	6	5	4	3	2	1(LSB)
dp	g	f	e	d	c	b	a

Table2 – Raw Data Standard Character Values

Symbol	Hex	Dec	Ascii
0	0x00	0	48
1	0x06	6	49
2	0x5B	91	50
3	0x4F	79	51
4	0x66	102	52
5	0x6D	109	53
6	0x7D	125	54
7	0x07	7	55
8	0x7F	127	56
9	0x67	103	57
A	0x77	119	65
B	0x7C	124	66
C	0x39	57	67
D	0x5E	94	68
E	0x79	121	69
F	0x71	113	70

4.2 Auditory Indicator (Optional)

The user can program the auditory indicator frequency, duty cycle and repeat value. The frequency is programmed by setting the buzzer's frequency time variable, the units are in 256us. For example a desired buzzer frequency of 1KHz would yield a frequency value of around 4. The buzzer's ontime and offtime variables are used to program the duty cycle of the buzzer. These units are in 50ms. So if you wanted the buzzer to turn on and off every second you would program 10 for the ontime and offtime. The repeat value dictates what mode the buzzer will be in. If a value of zero is used for the repeat value then the buzzer will sound continuously at the frequency specified until the user turns it off. If a value of 255 is used then the buzzer will sound at the frequency and duty cycle specified until the user turns it off. If any other value is used the buzzer will sound at the frequency and duty cycle specified for the repeat value number of times.
To increase the buzzer volume a small hole may be cut under the piezo buzzer in the plastic case.

5.0 Programming

5.1 Programming Overview

There are two ways to communicate to the USB device. They are the Direct Method and the DLL Method. The DLL method is the easier of the two to use. The DLL method adds slightly more processor overhead but relinquishes the programmer from having to deal with the low level commands of the USB device. The following sections describe how to communicate to the Delcom USB Numeric Display device. Please use this document along with the sample code available on the distribution disk and Delcom web site.

Requirements

Delcom USB Numeric Display Device
USB ready computer running Win98, ME, Win 2000, or XP.
Microsoft Visual C++ version 4.0 or better, or equivalent 32 bit compiler for x86 based system, or Visual basic compiler.
Knowledge of C/C++ or Visual basic.

5.2 Programming with the DLL

Please see the DelcomDLL documentation available on the distribution disk and website for more information. Below is a simple example in C using the DelcomDLL.

```
#include "stdafx.h"
#include "DelcomDLL.h"
int main(int argc, char* argv[])
{
char DeviceName[MaxDeviceLen];
HANDLE hUsb;
DelcomGetNthDevice(USBNDSPY, 0, DeviceName); // get devicename
hUsb = DelcomOpenDevice(DeviceName,0);       // open device
DelcomNumericMode(hUsb,ON);           // Display On
DelcomNumericAuto(hUsb,1234.56);      // Send Data
DelcomCloseDevice(hUsb);              // close device
return(0);
}
```

5.3 Programming with Direct Method

To communicate with USB Numeric Display 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 DeviceIOControl() to communicate to the USB Visual Signal Device and CloseHandle() to close it. To send commands to the USB Visual Signal device, simply build a command packet and submit it using the DeviceIOControl function.

Device Enumeration:
In order to communicate to the USB device one must first find its device name. The device name consists of a string of number representing a physical port plus the GUID (global unique identifier) for the device. In XP the PID (Product Identification Number) and the VID (Vendor Identification Number) are also include in this string. The device name can change each time you plug in an additional device or plug the device into a different USB port or hub on your computer system.

The GUID for the Delcom USB Numeric Display device is {5BAFC627-C81C-4832-9AAA-C5A60B25715D}, and a typical complete device name looks like \\.\0000000000000012#{5BAFC627-C81C-4832-9AAA-C5A60B25715D}.

Device Name Registry Method
There are two ways to get the device name. The easiest method is to read the device name from the registry. When a USB Numeric Display 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. Then the user just reads the name out of the registry. This method has one disadvantage. It can't be used when more than one USB Visual Signal device is plugged in to your computer, because only the last device name will be 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\Delcom\USBNDSPY\Parameters\

You can uses regedit.exe to find the entry. It is also a good place to copy the GUID from.

VB Registry Example

Here is an example in Visual Basic on how to read the device name from the registry.

```
DIM DeviceName as string
DeviceName = GetRegValue(HKEY_LOCAL_MACHINE, _
            "System\CurrentControlSet\Services\Delcom\USBNDSPY\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
```

Device Name Enumeration Method

The second method to get the device name is to use Windows™ device manager. To do this one calls a function in the setupapi.dll. Simply poll the device manger with the USB Visual Signal GUID for all the devices that match the GUID given. The device manager will return the device names for all the devices currently available on your system. This is the better way of getting the device name. It allows the user to use multiple devices on the same computer. The disadvantage is that it is a little more complicated.

C Enumeration Example

Below is a C example using this enumeration method.

Use the DEFINE_GUID macro to build the GUID.

```
// {5BAFC627-C81C-4832-9AAA-C5A60B25715D}
DEFINE_GUID(USBNDSPY_GUID,
      0x5bafc627, 0xc81c, 0x4832, 0x9a, 0xaa, 0xc5, 0xa6, 0xb,
      0x25, 0x71, 0x5d);
```

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

```
HDEVINFO hinfo = SetupDiGetClassDevs(&USBNDSPY_GUID, 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 USB Visual Signal device.

Once you have a handle to the device information set, you can perform an enumeration of all the devices that export the particular interface you're interested in. See Microsoft function documentation for more information on setupapi.dll 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)
          &USBNDSPY_GUID,i, &Interface_Info))
         {
         SetupDiDestroyDeviceInfoList(hInfo);
         return(i);
         }
```

```
      DWORD needed;    // get the required length
      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 USB Numeric Display device that is plugged into your computer. If you know that you will only support one USB device 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 have a port number prefixed to the original GUID. The port number is related to the installation order of the plug and play devices on your machine and cannot be predetermined. The device name should look like the following.

```
\\.\0000000000000012#{5BAFC627-C81C-4832-9AAA-C5A60B25715D}
```

This is the complete device name one will use in order to communicate with the USB Numeric Display device.

Device Communications:

Open Device

To begin communicating with the USB Numeric Display device 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, un-powered, or opened by another program this function will fail.

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

Device Close

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.*

```
CloseHandle( hUsbDevice ) ;
```

Device Communications
Device I/O Control

The DeviceIOControl() function provides the communication method between the users and the device. This function accepts CTL_CODES and users buffers that are passed to the device driver and eventually the USB device.

```
success = DeviceIoControl(hUsbDevice,
        IOCTL_USBIODS_SEND_PACKET,
        &TxPacket, TxPacketLen,
        &RxPacket, RxPacketLen,
        &nBytes, NULL);
```

The CTL Codes are predefined codes that describe the desired action to take place. There are many different codes but for our purposes we are only concerned with the send packet code.

Below is the CTL_CODE generation shown in C.

```
#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 DELCOM_USBIO_IOCTL_VENDOR 0x0800          // Vendor defined

#define IOCTL_USBIO_WRITE_PACKET     CTL_CODE(FILE_DEVICE_UNKNOWN,  \
                                     DELCOM_USBIO_IOCTL_VENDOR+10,\
                                     METHOD_BUFFERED,  \
                                     FILE_ANY_ACCESS)
```

The above code generates a CTL_CODE of 0x222028. You can just use this number instead for using the above code, see below. For VB code use &H222028.

```
#define IOCTL_USBIO_SEND_PACKET 0x222028    // for C
Const CTL_CODE_SEND_PACKET = &H222028       '  for VB
```

Once you have the CTL_CODE the next step is to make the command packet. This is a simple structure in which you just set the fields for a particular command. The fields in the command packet are described in the Direct Methods Commands below. Simply fill the structure and send it to the USB device with the DeviceIOControl function. For read commands the DeviceIOControl function returns the data in the RxPacket. The length of the sent packet is 8 to 16 bytes and the receive packet is always 8 bytes long.
The packet command structure consists of the following elements.

```
// Command Packet Structure define in C
typedef struct _ioPacket{
     unsigned char  Recipient;
     unsigned char  DeviceModel;
     unsigned char MajorCmd;
     unsigned char MinorCmd;
     unsigned char DataLSB;
     unsigned char DataMSB;
     unsigned short Length;    // length of ExtDate
     unsigned char ExtData[8];
} VENDORPACKET,*PVENDORPACKET;

‘ Command Packet Structure define in VB
Public Type PacketStructure
    Recipient   As Byte
    DeviceModel As Byte
    MajorCmd    As Byte
    MinorCmd    As Byte
    DataLSB     As Byte
    DataMSB     As Byte
    Length      As Integer      ‘length of ExtDate
    ExtData(8)  As Byte
End Type
```

C Example

This C code example sends the packet and receives the data in the same packet that was sent to it. On error it returns –1.

```
int UsbIoCtrl(PVENDORPACKET pPacket)
      {
      ULONG nBytes;
      BOOLEAN Success;

      Success = DeviceIoControl(hUsb,
            IOCTL_USBIO_WRITE_PACKET,
            pPacket, 8+pPacket->Length, pPacket, 8, &nBytes,
            NULL) ;

      if(!Success ) //|| ( nBytes != sizeof(VENDORPACKET) ) )
            {
            if(Verbose)MessagePopup ("UsbIoCtrl
                  Error","DeviceIoControl call failed!");
            return(-1);
            }
      else
            return(0);
      }
```

VB Example

This VB code example sends the packet and receives the data in the returned value of the function.

```
'Sends the USB packet to the device
Function SendPacket(ByRef TxPacket As PacketStructure) As PacketStructure
Dim lpResult As Long
Dim RxPacket As PacketStructure

TxPacket.Recipient = 8      ' always 8
TxPacket.DeviceModel = 18   ' always 18
TxPacket.LengthLSB = 0      ‘ always 0
TxPacket.LengthMSB = 0      ‘ always 0

If 0 = DeviceIoControl(hDevice, CTL_CODE_SEND_PACKET, TxPacket, 8, _
                       RxPacket, 8, lpResult, 0) Then
    MsgBox "SendPacket() DeviceIoControl Failed."
    Exit Function
End If

SendPacket = RxPacket

End Function
```

Registry Keys:

The following is a list of registry keys that the USB I/O driver adds to the registry. To access the registry run RegEdit.exe. The registry keys are located at:

```
HKEY_LOCAL_MACHINE\System\CurrentControlSet\Services\Delcom\USBNDSPY\Parameters
```

Delcom USB Visual Signal Registry Keys

- DebugLevel Used for debugging should always be zero.
- BootUpTest Used for testing should always be zero.
- DeviceName This string contains the device name of the last USB IO device loaded.

5.4 Direct Method Commands.

All commands are passed to the USB device in a command packet. The command is filled and sent to the USB device using the DeviceIOControl Windows™ command. All command packets are at least 8 byte long, maximum of 16 bytes and all receive data is 8 bytes long.

Direct Command Packet Format:

Recipient	Byte	Always 8 for the USB IO device.
Device Model	Byte	Always 18 for the USB IO device
Major Command	Byte	See Below
Minor Command	Byte	See Below
Data LSB	Byte	See Below
Data MSB	Byte	See Below
Length	Short (2 Bytes) Length of DataExtension.	
DataExtension	0-8 Bytes – (Only use by certain commands).	

Command Number		Data		v e r
Major	**Minor**	**Length**	**Command Description**	
10	-	-	**WRITE FUNCTIONS**	
10	0	0	*Dummy command.* Does nothing, used for testing.	
10	1	0	*Writes the LSB to port 0.* Port 0 is defaulted high after reset.	
10	2	0	*Writes the LSB to port 1.* Port 1 is defaulted high after reset.	
10	10	0	*Writes the LSB to port 0 and the MSB to port* 1.	
10	11	0	*Sets or resets the port 0 pins individually*. The LSB resets the corresponding port pin(s) and the MSB sets the corresponding port pin(s) on port 0. Resetting the port pin(s) takes precedence over setting the bits.	
10	12	0	*Sets or resets the port 1 pins individually*. The LSB resets the corresponding port pin(s) and the MSB sets the corresponding port pin(s) on port 1. Resetting the port pin(s) takes precedence over setting the bits.	
10	80	0	*Display On/Off.* Controls weather the LEDs are either on or off. A value of zero in the DataLSB turns the device off. A non-zero value in DataLSB turns on the device. On power up the device is off by default.	
10	81	0	*Display Flash Control.* Controls the flash rate of the display. The flash rate is passed in the DataLSB variable. The valid range is 0-255, where 0 equal flash off and is the default value. Units are in 10ms.	
10	82	0	*Display Setup.* This command is used to setup the number of digits used by the scan algorithm. The DataLSB is the scan mask, each bit in the mask corresponding to the mask pin. The LSB bit is first digit. The DataMSB is the number of digit present. For example to setup the display for 6 digits one must set the DataLSB to 0x3F and the DataMSB to 6. For an 8 digit display one must set DataLSB to 0xFF and DataMSB to 8. The default value of DataLSB is 0x3F and DataMSB is 6. When scan pin aren't being used they can be used as I/O.	
10	85	1-8	Write Data Command. This functions writes the data in the data extension on the Led display. Data Extension 0 is write to digit 0 and data extension 1 is written to digit 1 and so on. The DataLSB controls the mode to use. A DataLSB value of 0 is Raw Mode, 1 is Hex mode and 2 is Ascii Mode. The Data Extension length must be set to the number of digits you want to display. All the remaining digit are always cleared. For instance if the Data Extension Length is set to 4 then only the data in the first four data extension bytes will be written to the display. Digits 5-8 will be cleared. Valid range for data extension is 0-255 in raw mode, 0-15 in Hex mode and '0'-'9','A'-'F','a'-'f' in Ascii mode. Value outside this range will be set to hyphen '-'.	
10	86	0	*Set Decimal Point.* This functions sets up the decimal point. The DataLSB value is used set the decimal point function. Each bit in the DataLSB correspondes to a decimal point. The LSB bit is the decimal point of the first digit. A one (high) in the DataLSB will turn the decimal point on. Default value is zero, all decimal points off. For example if DataLSB is set to 0x09, decimal points on digits one and four be turned on, 0xFF will turn all decimal points on.	
10	70	3	*Buzzer Setup.* This command setup the buzzer. There are 5 variable passed with this command. The LSB byte controls the on and off function of the buzzer. A zero value in the LSB will turn the buzzer off. A value of one will turn the buzzer on using the other 4 variables. The MSB byte contains the frequency value, its units are in 256us. The ExtData[0] byte contains the repeat value. A repeat value of zero places the buzzer in continuous 100% duty cycle mode. A repeat value of 255 places the buzzer a continuous variable duty cycle defined by the on/off time below. Any other repeat value will places the buzzer in a duty cycle mode for the number of repeat time specified. The ExData[1] byte holds the on time duty cycle variable and the ExData[2] byte holds the off time duty cycle variable.	

Read Commands

Command Number		Data		ver
Major	**Minor**	**Length**	**Command Description**	
11	-	-	**READ FUNCTIONS** All read functions return 8 bytes. See individual commands for format.	
11	0	0	*Read ports 0 and port 1*. The first byte (LSB) will contain the current value on port 0 and the second byte (MSB) will contain the current value on port 1. This command can be used to read the current LED and button state.	
11	8	0	*Reads the button event counter value.* This command returns the 4 byte event counter value and then resets the counter. If the counter over flows then the over flow status byte will be set to 0xFF otherwise it will be 0x0. The event counter is returned in the first 4 bytes and the over flow byte is in the 5 byte.	
11	9	0	*Reads system variables.* This function returns the following system variables. Byte0: Control Register. Byte1: Clock Generator Pre-Scalar. Byte4: USB Port Address. The control register has the following bits. Bit4: Set when buzzer is running. Does not include duty cycle off time. Bit5-7: Spare	
11	10	0	*Reads the firmware information.* Byte 0-3: Unique Device Serial Number. DWORD Little Endian. Byte 4: Firmware Version. Byte 5: Firmware Date. Byte 6: Firmware Month. Byte 7: Firmware Year.	

6.0 Disk Contents

[Root- Directory]
USBDELVI.INF - The installation file for Visual Indicator
USBDELVI.SYS - The Visual Indicator Driver

[Documents- Directory]
USBDELVI.pdf – Visual Indicator Development Manual
AN201.pdf – USB Windows™ Installation

[C-Code – Directory]
A simply MFC C++ Applications and Sample Code

[VB-Code – Directory]
A simply VB Code Applications and Sample Code

7.0 Release Versions

7.1 Firmware Version

Version 1.0 - Initial Release

7.2 Driver Version

Version 1.00.5001.4 - Initial Release

8.0 Ordering Information

Part Number	Description
806004	USB 4 Digit Numeric Display Red
806006	USB 6 Digit Numeric Display Red
806008	USB 8 Digit Numeric Display Red
806014	USB 4 Digit Numeric Display Red w/Enclosure
806016	USB 6 Digit Numeric Display Red w/Enclosure
806018	USB 8 Digit Numeric Display Red w/Enclosure
806005	USB 4 Digit Numeric Display Green
806007	USB 6 Digit Numeric Display Green
806009	USB 8 Digit Numeric Display Green
806015	USB 4 Digit Numeric Display Green w/Enclosure
806017	USB 6 Digit Numeric Display Green w/Enclosure
806019	USB 8 Digit Numeric Display Green w/Enclosure

9.0 Notes

9.1 Using USB extension cables.

An USB extension cable can be used with device but one must make sure the voltage at the device does not drop below 4.0 volts. If the voltage at the device drops below 4.0 volts the device will reset. For cable lengths longer than 15feet powered USB should be used.

9.2 Using USB Hubs

An USB hub can be used with this device but one must make sure that the hub is a full powered hub. That can supplies at 500mA to each port. Some low-end hubs provide no more than 100mA of current to each port. If the hub does not supply the required current the device will reset.

Appendix A: Related Documents and Web sites

Delcom Engineering Web Site → www.delcom-eng.com
Universal Serial Bus Specification → www.usb.org
Microsoft Development Network → www.msdn.microsoft.com