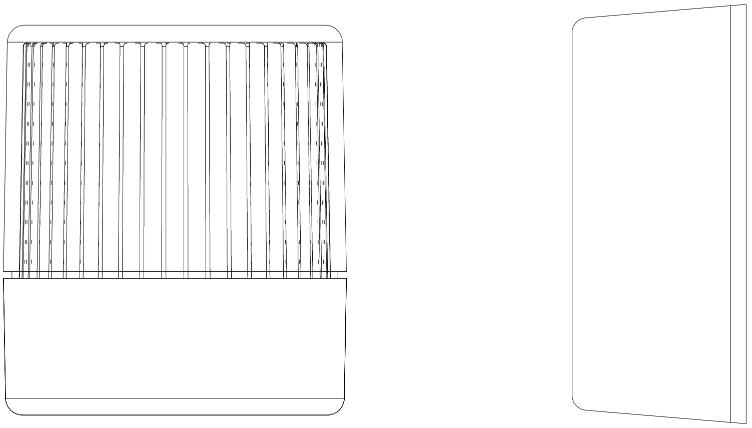
USB Visual Indicator Development Manual



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

Document Ver USBVIDM 1.2 March 24, 2004

Contents

1.0 Introduction

This manual describes both Multi-Color and Single Color USB Visual Indicators. The USB Visual Indicator is a visual and optional auditory signal device powered and activated via the USB port. The visual signal is produced by LEDs in up to three colors. Available colors are Green, Red, Yellow, Orange and Blue. LED 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 also contains a optional momentary button that can be user 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 Visual Indicator, 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 Visual Indicator distribution disk in the floppy drive and select that drive to install the driver. This procedure for loading of 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 Visual Indicator 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 Visual Indicator device is compatible with the following operating systems. Windows 98, ME, 2000, XP and MAC OSX.

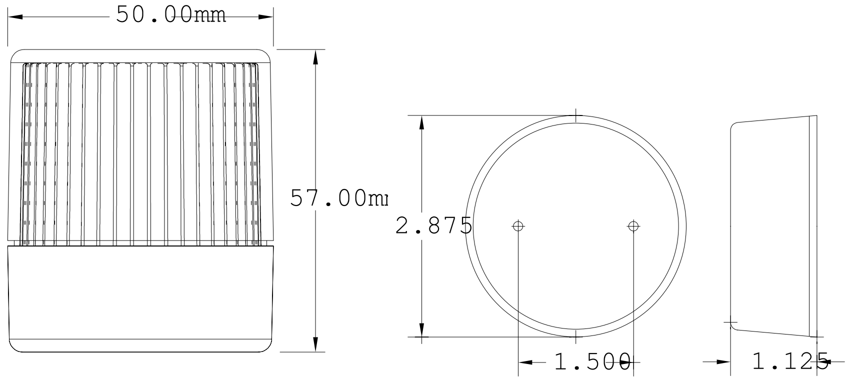
3.2 Mounting

The Multi-Color Visual Indicator comes with two mounting options. These are surface mount and pole mount. For flat surface mounting either use as a stand alone or with the Velcro button included. For elevated mounting, vertical mounting, or high vibration mounting use the pole mounting method. The pole mounting method uses a standard ¼" NPT pipe (OD 0.540") , the device can accommodate a ¼" NPT or M14 tread.

The Single Color Visual Indicator can be mounted on any flat surface. The device has two mounting holes at 1.5 inch spacing.

3.2 Mechanical

The USB Visual Indicators has the following dimensions.



3.3 Electrical

The USB Visual Indicator consumes 8mA when all LEDs and buzzer is off. 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 and produce a typical light output of 16,000 mcd. With version 8 of the USB light each color can be variable in intensity. Variable range is from 0 to 100%, default power up value is 80%. Running all colors on at the same time at 100% is not recommended, because you can exceed the allowable current draw from the USB port. This is especially true with the Red, Green and Blue combination. You can run all colors on at about 80% at the same time, provided you have a USB compliant port capable of supplying 500ma. If you exceed the current limit the device will automatically turn off and you will have to reset the device. The table below indicates typical current consumptions.

Color\Pwr	5%	25%	50%	75%	100%
Red	2mA	19mA	51mA	84mA	134mA
Yellow	2mA	19mA	51mA	84mA	134mA
Blue	2mA	29mA	81mA	130mA	200mA
Green	3mA	34mA	95mA	150mA	225mA

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 Visual Indicators

The LED visual indicators can be programmed in three modes. On, Off and Flashing Mode. To setup the flashing mode one simply programs the ontime and offtime variables. The ontime and offtime have units that are equal to 1.024ms times the prescalar value. The default value for the prescalar is 10. For example if you wanted to produce a flash rate of 10Hz at 50% duty cycle with the default prescalar value of 10 you would program the ontime with 5 and off time with 5. The flash mode also has a synchronies function. This function allows all three LEDs to be synchronized with each other. The command uses an offset value that has units of the 1.024ms times the prescalar value. Using the offset value you can delay when the LED will start its flash mode. The offset default value is zero and will cause all three to start flashing at the same time.

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 bottom of the plastic base.

4.3 Input Button (Optional)

The input button consists of a momentary button that is actuated by press the top lens piece into the base piece. The button status can be read to trigger user events. The button can also be placed into two internal modes, Auto Clear and Auto Confirm. When Auto Clear is enabled the device will clear the current visual and auditory conditions. When Auto Confirm is enabled the device will respond with two quick tones each time the button is pressed.

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 Signal device. The following sections describe how to communicate to the Delcom USB Visual Signal device. Please use this document along with the sample code available on the distribution disk and Delcom web site.

Requirements

Delcom USB Visual Signal Device
Delcom Visual Signal USB Driver
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(USBIODS, 0, DeviceName);  // get the devicename
hUsb = DelcomOpenDevice(DeviceName,0);       // open device
DelcomLEDControl(hUsb,BLUELED, 2);           // flash led
DelcomCloseDevice(hUsb);              // close device
return(0);
}
```

5.3 Programming with Direct Method

To communicate with USB Visual Signal 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 Visual Signal device is {59BD73A6-822E-4684-9530-0754FE897113}, and a typical complete device name looks like
\\.\0000000000000012#{59BD73A6-822E-4684-9530-0754FE897113}.

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 Visual Signal 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\USBDELVI\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\USBDELVI\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.

```
// {59BD73A6-822E-4684-9530-0754FE897113}
DEFINE_GUID(LAMP_GUID,
      0x59bd73a6, 0x822e, 0x4684, 0x95, 0x30, 0x7, 0x54, 0xfe,
0x89, 0x71, 0x13);
```

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(&LAMP_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)
          &USBIODS_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 Visual Signal device that is plugged into your computer. If you know that you will only support one USB I/O 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#{59BD73A6-822E-4684-9530-0754FE897113}
```

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

Device Communications:

Open Device

To begin communicating with the USB Visual Signal 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;
     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
    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\US
BDELVI\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**		**ver**
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. Port 0 has the following connected to it. P0.0 – Momentary Button Switch. This is an input and should be left high (1).	
10	2	0	*Writes the LSB to port 1.* Port 1 is defaulted high after reset. Port 1 has the following connected to it. P1.0 – Green LED, P1.1 – Red LED, P1.2 – Blue LED, P1.3 – Buzzer LEDs are active low (0). A low (0) will turns the LED on.	
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	19	0	*Loads the Flash Mode Clock Generator Global Pre-scalar value.* Default value is 10, range = 1 to 255. Increasing this number decreases all the clock function frequencies. This value can be read with command 11-9.	
10	20	0	*Enables or disables Flash Mode on port 1*. The lower nibble of the LSB disables the corresponding port pin(s) and the lower nibble of the MSB enables the corresponding port pin(s). Disabling the port pin(s) takes precedence over enabling. i.e. LSB=0x00 and MSB=0x02 would enable flash mode on the Red LED and a LSB=0x02 and MSB= 0x00 would disable the Red flash mode.	
10	21	0	*Loads the frequency and duty cycle for pin 1.0* Green LED. See format below.	
10	22	0	*Loads the frequency and duty cycle for pin 1.1* Red LED. See format below.	
10	23	0	*Loads the frequency and duty cycle for pin 1.2* Blue LED. See format below.	
			Frequency and duty format. The LSB sets the period when the port pin is high and the MSB sets the period when the port pin is low. The resolution of the period is 1.024ms times the prescalar value. The resolution of the duty cycle is 0.39 percent. Clock pins can be preset to a predefined state	
10	25	0	*Synchronizes the clock generation.* This command synchronizes all the LED clock generators to start at an initial phase delay, see below. Initial phase delay resolution is in 1.024ms times the prescalar. Initial phase delay registers below are clear after this command is sent. The lower nibble of the LSB enables this function on each corresponding pin. The lower nibble of the MSB presets the initial output value on the corresponding pin to on or off. Initial phase delay is also referred to as the offset value in other documents. i.e. LSB=0x01 and MSB=0x03 will synchronizes the Green and Red LEDs. The Green will start in the off state and the Red LED will start in the on state. If the initial phase delay for any LED was non-zero the Flash mode would lag for the time specified.	
10	26	0	*Load initial phase delay on pin 1.0, Green LED*. See Synchronies function above.	
10	27	0	*Load initial phase delay on pin 1.1, Red LED*. See Synchronies function above.	
10	28	0	*Load initial phase delay on pin1.2, Blue LED*. See Synchronies function above.	
10	34	0	*Load light intensity* This command sets the LED color power level. The LSB byte selects the LED, 0=green, 1=red and 2=blue/yellow. The MSB byte sets the power level, where 0 is the dimmest and 100 is the brightest. Default power up value is 80. See Section 3.4.	8
10	38	0	*Enable/Disable Events Counter.* This command sets up the event counter. LSB data byte enables this function on the corresponding pin on port 0. The MSB data byte disabled this function on the corresponding pin on port 0. Once enabled the system will count events on the enabled pin on the active edge, high to low. The	

			event counter value is read with command 11-8. This feature is off by default. This command may be used to count the number of times the button on pin 0.0 has been pressed. To enable set LSB=0x01 and MSB=0x00.	
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.	7
10	72		*Button Setup.* This command setup the button modes. The two button modes are Auto Clear and Auto Confirm. These modes may be used together or individually. The LSB byte clears the corresponding bit and MSB byte sets the corresponding bit. The Auto Clear function is controlled with bit 6 and the Auto Confirm function is controlled with bit 7. i.e. LSB =0x00 and MSB=0x03 will enable both modes. Auto Clear mode will place all the LEDs and the buzzer in the off state. The Auto Confirm will cause the buzzer to beep twice when button is pressed. To use either of these two modes you must first enable the event counter on pin 0.0. see command 10-38.	7

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: Set when button event counter overflows. Bit6: Set when Auto Clear mode is on. Bit7: Set when Auto Confirm mode is on.	
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

[USBSignal – Directory]
A fully featured MFC C++ Applications and Sample Code

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

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

[DosCode – Directory]
Dos Utility Applications and Sample Code

7.0 Release Versions

7.1 Firmware Version

Version 7.0	- Initial Release
Version 8.0 03/24/04	- Added variable light intensity feature.

7.2 Driver Version

Version 1.00.5001.4	- Initial Release
Version 1.00.5001.5	- Fixed blue screen bug, related to heavy polling.
Version 1.00.5001.6	- Fixed fatal error on Win2000 when power button/sleep mode was entered.

8.0 Ordering Information

Part Number	Description
804000	USB RBG Visual Signal Indicator
804002	USB RYG Visual Signal Indicator
804010	USB RBG Visual Signal Indicator Pole Mount
804012	USB RYG Visual Signal Indicator Pole Mount
804402	USB Red Visual Signal Indicator
804403	USB AmberVisual Signal Indicator

9.0 Notes

9.1 Using USB extension cables.

An USB extension cable (up to 15ft) can be used with this 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.

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 or bus powered 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