

CollectData.frm

```
Private Sub Form_Load()  
' This subroutine runs as soon as the program starts (ie as soon as this FORM is loaded)  
' Initialize my global variables  
ConnectionStatus(0) = "No device"  
ConnectionStatus(1) = "Device connected"  
ConnectionStatus(2) = "Device FAILED enumeration"  
ConnectionStatus(3) = "Device general FAILURE"  
ConnectionStatus(4) = "Device caused overcurrent"  
ConnectionStatus(5) = "Not enough power for device"  
  
' Initialize the display  
For i& = 0 To 3: HCD(i&).BackColor = RGB(256, 0, 0): Next i& 'Red = start  
StatusBox.Text = "Searching for Host Controllers"  
Collect_Data.Height = 1725  
  
' Descriptors will be displayed in a different window, load it  
Load Display_Descriptors  
  
' Look for Host Controllers.  
' I limit the search to 3. There may be more but this is unlikely.  
' The Host Controller Buttons are HCD(0) to HCD(3)  
' Try opening the controller using it's Symbollic Name  
,  
,  
  
For ControllerIndex& = 0 To 3  
HostControllerName$ = "\\.\HCD" & ControllerIndex&  
Dim SA As Security_Attributes  
HostControllerHandle& = CreateFile(HostControllerName$, &HC0000000, 3, SA, 3, 0, 0)  
If HostControllerHandle& > 0 Then  
HCD(ControllerIndex&).Tag = HostControllerHandle&  
HCD(ControllerIndex&).BackColor = RGB(0, 256, 0) 'Green = GO  
HCD(ControllerIndex&).Enabled = True  
Else  
HCD(ControllerIndex&).BackColor = RGB(256, 128, 0) 'Amber = wait  
HCD(ControllerIndex&).Enabled = False  
End If  
Next ControllerIndex&  
StatusBox.Text = "Select a Host Controller"  
End Sub  
  
Private Sub HCD_Click(index%)  
' Can only click HCD buttons with a Host Controller behind them  
' Host controller handle is stored in the button's TAG field  
,  
,  
' Get the name of the host controller  
HostController$ = GetNameOf("Host Controller", HCD(index).Tag, &H220424)  
StatusBox.Text = "Host Controller: " & HostController$ & " selected"  
Device_Display.Clear  
Collect_Data.Height = 6900  
,  
,  
' Get the name of the Root Hub and open a connection to it  
RootHubName$ = GetNameOf("Root Hub", HCD(index).Tag, &H220408)  
RootHubHandle& = OpenConnection(RootHubName$)  
,  
,  
' Get the node connection information.  
' **This is commented out, it should work but doesn't. Assume Root Hub has a Device ID of 1  
'Status& = DeviceIoControl(RootHubHandle&, &H22040C, RootHubNodeConnection.ConnectionIndex, 256,  
RootHubNodeConnection.ConnectionIndex, 256, BytesReturned&, 0)  
'If Status& = 0 Then ErrorExit ("Could not get connection information from Root Hub")  
,  
Call GetNodeInformation(RootHubHandle&)  
,  
,  
' Save this information in our data table  
DeviceData(0).DeviceHandle = RootHubHandle&  
DeviceData(0).DeviceType = 1 'Root Hub  
Device_Display.AddItem "001 : Root Hub"  
,  
,  
' Discover what is connected to the ports of this Root Hub  
Level& = 0  
Level& = GetPortData(RootHubHandle&, DeviceData(0).NodeData.NodeDescriptor.PortCount, Level& + 1)  
  
StatusBox.Text = "Select a device, then choose a descriptor to display"  
End Sub  
  
Function GetPortData(Handle&, PortCount As Byte, HubDepth&) As Long  
Dim ThisDevice As Byte  
  
For PortIndex& = 1 To PortCount  
Call GetNodeConnectionData(Handle&, PortIndex&)  
  
ThisDevice = 0 ' default value, no device connected  
PortStatus& = DeviceData(DataIndex).ConnectionData.ThisConnectionStatus(0) ' save some typing!  
If PortStatus& = 1 Then  
ThisDevice = DeviceData(DataIndex).ConnectionData.DeviceAddress(0)  
DeviceData(DataIndex).DeviceHandle = Handle&  
End If  
' Create an indented display so that Hubs and their connections are easily seen  
Indent$ = " ": For i& = 1 To HubDepth: Indent$ = Indent$ & " ": Next i&  
DeviceName$ = ThreeDecimalCharacters$(ThisDevice) & Indent$ & " Port["
```

```

Mid$(DeviceName$, 10) = ":"

If PortStatus <> 1 Then ' There is not a valid device on this port, tell user
    Device_Display.AddItem DeviceName$ & PortIndex & "]" = " & ConnectionStatus$(PortStatus&)
Else ' have a Device or a Hub connected to this port

    If DeviceData(DataIndex).ConnectionData.DeviceIsHub Then
    '
    ' Need to discover how many ports are supported on this hub.
    ' Follow the same procedure as we did for the root hub = get it's name, "open" it and get the node information
        ExternalHubName$ = GetExternalHubName(PortIndex&, Handle&)
        ExternalHubHandle& = OpenConnection(ExternalHubName$)
        Call GetNodeInformation(ExternalHubHandle&)
        DeviceData(DataIndex).DeviceType = 2 'Hub
    ' LAST thing we do is update the display status of this device connection
        Device_Display.AddItem DeviceName$ & PortIndex & "]" = Hub Connected"
    '
    ' Discover what, if anything, is connected to the ports of this Root Hub
        Level& = GetPortData(ExternalHubHandle&, DeviceData(DataIndex - 1).NodeData.NodeDescriptor.PortCount, HubDepth& + 1)

    Else 'we have a device connected to this port
        DeviceData(DataIndex).DeviceType = 3 'IODevice
        Device_Display.AddItem DeviceName$ & PortIndex & "]" = IO Device Connected"
    End If 'USBDeviceInfo.DeviceIsHub
    End If 'PortStatus <> 1
Next PortIndex&
End Function

Private Sub Device_Display_Click()
' User has selected a device
Selected& = Device_Display.ListIndex
Entry$ = Device_Display.List(Selected)
DeviceID& = Val(Left$(Entry$, 3))
If DeviceID& = 0 Then
    StatusBox.Text = "There is no device connected to this node, please choose another"
Else
    StatusBox.Text = "Fetching descriptors"
    Call CollectDescriptors(Selected&)
    Call Display_Descriptors.Initialize
End If
End Sub

Private Sub CollectDescriptors(Selected&)
' Collect all of the descriptors from the selected device and store them in the DescriptorData byte array
' Start with the Device Descriptor
For i& = 1 To 18: DescriptorData(i&) = DeviceData(Selected&).ConnectionData.ThisDevice.Contents(i& - 1): Next i&
Next i& = 18
' Now get local copies of some key variables
Dim Configuration As Byte: Dim StringIndex As Byte
Handle& = DeviceData(Selected&).DeviceHandle
ConnectionIndex& = DeviceData(Selected&).ConnectionData.ConnectionIndex
ConfigurationCount = DeviceData(Selected&).ConnectionData.ThisDevice.Contents(17)
For Configuration = 1 To ConfigurationCount
    TotalLength& = GetConfigurationDescriptor(Handle&, ConnectionIndex&, Configuration - 1)
' Copy the Configuration Descriptor into the common data buffer
    For i& = 1 To TotalLength&: DescriptorData(Nexti& + i&) = PCHostRequest.ConfigurationDescriptor(i& - 1): Next i&
    Nexti& = Nexti& + TotalLength&: Next Configuration
' Check for Strings
StringIndex = 0
Do While TotalLength& <> 0
    TotalLength = GetStringDescriptor(Handle&, ConnectionIndex&, StringIndex)
    StringIndex = StringIndex + 1
    For i& = 1 To TotalLength&: DescriptorData(Nexti& + i&) = PCHostRequest.ConfigurationDescriptor(i& - 1): Next i&
    Nexti& = Nexti& + TotalLength&: Loop
End Sub

Private Function GetStringDescriptor&(Handle&, ConnectionIndex&, StringIndex As Byte)
PCHostRequest.ConnectionIndex = ConnectionIndex
PCHostRequest.PacketData.wValueLo = StringIndex: PCHostRequest.PacketData.wValueHi = 3 ' = type
If StringIndex = 0 Then
    PCHostRequest.PacketData.wIndex = 0
    Else: PCHostRequest.PacketData.wIndex = &H409: End If ' This SHOULD be read from String 0
PCHostRequest.PacketData.wLength = 254 ' = Max string length
Status& = DeviceIoControl(Handle&, &H220410, PCHostRequest.ConnectionIndex, 286, PCHostRequest.ConnectionIndex, 286 BytesReturned&, 0)
If Status = 0 Then BytesReturned = 12 ' No string, so return TotalLength = 0
GetStringDescriptor& = BytesReturned& - 12
End Function

Private Function GetConfigurationDescriptor&(Handle&, ConnectionIndex&, ConfigurationID As Byte)
PCHostRequest.ConnectionIndex = ConnectionIndex
PCHostRequest.PacketData.wValueLo = ConfigurationID: PCHostRequest.PacketData.wValueHi = 2 ' = type
PCHostRequest.PacketData.wIndex = 0: PCHostRequest.PacketData.wLength = 9
' First read just the Configuration Descriptor to discover 'Total Length'
' Note 21 = 13(Size of PCHostRequest) + 8(Size of PacketData)
Status& = DeviceIoControl(Handle&, &H220410, PCHostRequest.ConnectionIndex, 21, PCHostRequest.ConnectionIndex, 21, BytesReturned&, 0)
If Status = 0 Then ErrorExit ("Could not get initial Configuration Data")
' Now read Configuration+Interface+Endpoint+Class
TotalLength = 256 * PCHostRequest.ConfigurationDescriptor(3) + PCHostRequest.ConfigurationDescriptor(2)
PCHostRequest.ConnectionIndex = ConnectionIndex
PCHostRequest.PacketData.wValueLo = ConfigurationID: PCHostRequest.PacketData.wValueHi = 2 ' = type

```

```

PCHostRequest.PacketData.wIndex = 0: PCHostRequest.PacketData.wLength = TotalLength
Status& = DeviceIoControl(Handle&, &H220410, PCHostRequest.ConnectionIndex, TotalLength + 13, PCHostRequest.ConnectionIndex,
TotalLength + 13, BytesReturned&, 0)
If Status = 0 Then ErrorExit ("Could not get complete Configuration Data")
If BytesReturned& > 2000 Then ErrorExit ("Buffer Overflow for Configuration Descriptor")
GetConfigurationDescriptor& = BytesReturned& - 12
End Function

```

Display.frm

```

Public Sub Initialize()
' Control is passed to this module once all of the Descriptor information has been collected
' This module displays the information in a form as similar as Chapter 3 as possible
Call ClearDisplay
Call ParseDescriptors
' Determine which Class Descriptor, if any, is in Column 5
Display_Descriptors.Show
End Sub

Private Sub ClearDisplay()
For i% = 0 To 4
Call DisplayNames(i%)
Choice(i%).Clear
Choice(i%).AddItem "Display Names"
Choice(i%).ListIndex = 0
Next i%
End Sub

Private Sub ParseDescriptors()
DeviceCount = 0: ConfigurationCount = 0: InterfaceCount = 0: EndpointCount = 0: ClassCount = 0
index& = 1
Do While DescriptorData(index&) <> 0
Select Case DescriptorData(index& + 1) ' What TYPE of descriptor is this?
Case 1
DeviceCount = DeviceCount + 1: Choice(0).AddItem "Display Values"
Choice(0).ItemData(Choice(0).ListCount - 1) = index&
Case 2
ConfigurationCount = ConfigurationCount + 1: Choice(1).AddItem "Configuration " & ConfigurationCount
Choice(1).ItemData(Choice(1).ListCount - 1) = index&
Case 3
If StringCount <> 0 Then Choice(5).AddItem "String " & TwoHexCharacters$(CByte(StringCount)) & " = " & GetString$(index&)
StringCount = StringCount + 1
Case 4
InterfaceCount = InterfaceCount + 1
Choice(2).AddItem "Interface " & ConfigurationCount & ":" & InterfaceCount
Choice(2).ItemData(Choice(2).ListCount - 1) = index&
Case 5
EndpointCount = EndpointCount + 1
Choice(3).AddItem "Endpoint " & ConfigurationCount & ":" & EndpointCount
Choice(3).ItemData(Choice(3).ListCount - 1) = index&
Case Else ' Must be a Class Descriptor
ClassCount = ClassCount + 1
Choice(4).AddItem "Class(" & TwoHexCharacters$(CByte(DescriptorData(index& + 1))) & ")" & ConfigurationCount & ":" & ClassCount
Choice(4).ItemData(Choice(4).ListCount - 1) = index&
End Select
index& = index& + DescriptorData(index&)
Loop
' Fill out the default data for the Descriptors
For i% = 0 To 4
If Choice(i%).ListCount = 1 Then ' descriptor type is not present, remove from display
Choice(i%).Visible = False: Descriptor(i%).Visible = False
Else ' fill with data
Choice(i%).ItemData(0) = Choice(i%).ItemData(1)
Call AddDescriptorData(i%, 0)
End If
Next i%
If StringCount = 0 Then Choice(5).Visible = False
End Sub

Private Function GetString$(index&) ' Extract UNICODE string from Descriptor
Length& = DescriptorData(index&) - 2
temp$ = "": For i& = 2 To Length& + 1 Step 2: temp$ = temp$ + Chr$(DescriptorData(index& + i&)): Next i&
GetString$ = temp$
End Function

Private Sub Descriptor_Click(DescriptorID%)
' If a user clicks on a TEXT entry then this text is replaced with it's current VALUE
Selection& = Descriptor(DescriptorID%).ListIndex
Descriptor(DescriptorID%).List(Selection&) = TwoHexCharacters(Descriptor(DescriptorID%).ItemData(Selection&))
End Sub

Private Sub Choice_Click(DescriptorID%)
' If a user changes the SELECTION then the corresponding Descriptor DISPLAY must be updated
If DescriptorID% <> 5 Then
Selection& = Choice(DescriptorID%).ListIndex
If Selection& = 0 Then
Call DisplayNames(DescriptorID%)
Call AddDescriptorData(DescriptorID%, 0)
Else
Call AddDescriptorData(DescriptorID%, Selection&)

```

```

        Call DisplayDescriptor(DescriptorID%, Selection&)
    End If 'Selection& = 0
End If 'DescriptorID% <> 5
End Sub

Public Sub AddDescriptorData(DescriptorID%, Selection&)
' This subroutine updates the ITEMDATA of DISPLAY with the corresponding VALUES
index& = Choice(DescriptorID%).ItemData(Selection&)
Length& = Descriptor(DescriptorID%).ListCount
For i& = 0 To Length& - 1
    Descriptor(DescriptorID%).ItemData(i&) = DescriptorData(index& + i&)
Next i&
End Sub

Public Sub DisplayDescriptor(DescriptorID%, Selection&)
' This subroutine replaces Descriptor TEXT with Descriptor VALUES
Length& = Descriptor(DescriptorID%).ListCount
For i& = 0 To Length& - 1
    Descriptor(DescriptorID%).List(i&) = TwoHexCharacters$(Descriptor(DescriptorID%).ItemData(i&))
Next i&
End Sub

Public Sub DisplayNames(DescriptorID%)
' This subroutine clears the Descriptor then copies TEXT to the DISPLAY
Descriptor(DescriptorID%).Clear
Descriptor(DescriptorID%).AddItem "        Length"
Descriptor(DescriptorID%).AddItem "        Type"
Select Case DescriptorID%
Case 0 'Device
    Descriptor(0).AddItem "        USB"
    Descriptor(0).AddItem "        Version"
    Descriptor(0).AddItem "        Class"
    Descriptor(0).AddItem "        SubClass"
    Descriptor(0).AddItem "        Protocol"
    Descriptor(0).AddItem "        EP0_Size"
    Descriptor(0).AddItem "        Vendor"
    Descriptor(0).AddItem "        ID"
    Descriptor(0).AddItem "        Product"
    Descriptor(0).AddItem "        ID"
    Descriptor(0).AddItem "        Version"
    Descriptor(0).AddItem "        Number"
    Descriptor(0).AddItem "        iManufacturer"
    Descriptor(0).AddItem "        iProductName"
    Descriptor(0).AddItem "        iSerial#"
    Descriptor(0).AddItem "        Configurations"
Case 1 'Configuration
    Descriptor(1).AddItem "        Total"
    Descriptor(1).AddItem "        Length"
    Descriptor(1).AddItem "        Interfaces"
    Descriptor(1).AddItem "        ThisConfig."
    Descriptor(1).AddItem "        ConfigName"
    Descriptor(1).AddItem "        Attributes"
    Descriptor(1).AddItem "        Max.Power"
Case 2 'Interface
    Descriptor(2).AddItem "        ThisInterface"
    Descriptor(2).AddItem "        Alternate"
    Descriptor(2).AddItem "        Endpoints"
    Descriptor(2).AddItem "        Class"
    Descriptor(2).AddItem "        SubClass"
    Descriptor(2).AddItem "        Protocol"
    Descriptor(2).AddItem "        InterfaceName"
Case 3 'Endpoint
    Descriptor(3).AddItem "        ThisEndpoint"
    Descriptor(3).AddItem "        Attributes"
    Descriptor(3).AddItem "        Max. Packet"
    Descriptor(3).AddItem "        Size"
    Descriptor(3).AddItem "        Polling Interval"
Case 4 ' Class. Display will depend upon which Class
'    x = Descriptor(4).ItemData(1) 'Type
'    Select Case x
'        Case 33 'HID
            Descriptor(4).AddItem "Param3"
            Descriptor(4).AddItem "Param4"
            Descriptor(4).AddItem "Param5"
            Descriptor(4).AddItem "Param6"
            Descriptor(4).AddItem "Param7"
            Descriptor(4).AddItem "Param8"
            Descriptor(4).AddItem "Param9"
'    End Select
End Select
End Sub

```

USBInterface.bas

```

'
' Declare all of the USB Data Structures
'
' Note that most of these Data Structures MUST be defined a BYTES.
' This prevents Visual Basic "helpfully" aligning variables on their natural byte boundaries.
' Little Endian is assumed. ie If Byte(3)= Long, Then byte(0) = LSB

```

```

'
Public Type UNameType: Length As Long: UnicodeName(256) As Byte: End Type
Public Type UNodeType: ConnectionIndex As Long: Length As Long: UnicodeName(256) As Byte: End Type

Public Type SetupPacket
    RequestType As Byte: Request As Byte: wValueLo As Byte: wValueHi As Byte: wIndex As Integer: wLength As Integer: End Type

Public Type DescriptorRequest
    ConnectionIndex As Long: PacketData As SetupPacket: ConfigurationDescriptor(2000) As Byte: End Type

Public Type DeviceDescriptor
    Contents(17) As Byte: End Type
' Defined as a Byte Array to make later data movement simpler
' Length As Byte: DescriptorType As Byte: USBSpec(1) As Byte: Class As Byte
' SubClass As Byte: Protocol As Byte: MaxEP0Size As Byte: VendorID(1) As Byte
' ProductID(1) As Byte: DeviceRevision(1) As Byte: ManufacturerStringIndex As Byte
' ProductStringIndex As Byte: SerialNumberStringIndex As Byte: ConfigurationCount As Byte: End Type

Public Type HubDescriptor
    Length As Byte: HubType As Byte: PortCount As Byte: Characteristics(1) As Byte
    PowerOn2Good As Byte: MaxCurrent As Byte: PowerMask(63) As Byte: End Type

Public Type EndPointDescriptor
    Length As Byte: DescriptorType As Byte: EndpointAddress As Byte
    Attributes As Byte: MaxPacketSize(1) As Byte: PollingInterval As Byte: End Type

Public Type NodeInformation
    NodeType As Long: NodeDescriptor As HubDescriptor: HubIsBusPowered As Byte: End Type

Public Type NodeConnectionInformation
    ConnectionIndex As Long: ThisDevice As DeviceDescriptor: CurrentConfiguration As Byte
    LowSpeed As Byte: DeviceIsHub As Byte: DeviceAddress(1) As Byte: NumberOfOpenEndpoints(3) As Byte
    ThisConnectionStatus(3) As Byte: MyEndpoints(29) As EndPointDescriptor: End Type

' I keep all of the IO Device information I collect in a big table
' Most USB installations will only fill part of this table
Public Type CollectedDeviceData
    DeviceType As Long: DeviceHandle As Long: ConnectionData As NodeConnectionInformation
    NodeData As NodeInformation: End Type
Public DeviceData(200) As CollectedDeviceData
'
' All Descriptors are concatenated here once a device is selected
Public DescriptorData(2000) As Byte
'
' I need to send Requests to USB devices
Public PHostRequest As DescriptorRequest
'
Public ConnectionStatus(6) As String
'
' Declare my support sub-routines
Public Function DataIndex()
' All writes to the DeviceData table are done to entry DataIndex
' Need to keep DeviceData and IODevice_Display in sync
DataIndex = Collect_Data.Device_Display.ListCount
End Function
Public Function OpenConnection(Name$)
Dim SA As Security_Attributes
Handle& = CreateFile("\\.\\" & Name$, &HC0000000, 3, SA, 3, 0, 0)
If Handle& = 0 Then ErrorExit ("Could not open a connection to " & Name$)
OpenConnection = Handle&
End Function

Public Sub GetNodeInformation(Handle&)
' Get the node information
Dim BytesReturned&, Status&
Status& = DeviceIoControl(Handle&, &H220408, DeviceData(DataIndex).NodeData.NodeType, 256, DeviceData(DataIndex).NodeData.NodeType,
256, BytesReturned&, 0)
If Status& = 0 Then ErrorExit ("Could not get node information")
If BytesReturned& > 256 Then ErrorExit ("DeviceIoControl returned >256 bytes of data")
End Sub

Public Sub GetNodeConnectionData(Handle&, PortIndex&)
Dim BytesReturned&, Status&
DeviceData(DataIndex).ConnectionData.ConnectionIndex = PortIndex&
Status& = DeviceIoControl(Handle&, &H22040C, DeviceData(DataIndex).ConnectionData.ConnectionIndex, 256,
DeviceData(DataIndex).ConnectionData.ConnectionIndex, 256, BytesReturned&, 0)
If Status& = 0 Then ErrorExit ("Could not get Node Connection Data")
If BytesReturned& > 256 Then ErrorExit ("DeviceIoControl returned >256 bytes of data")
End Sub

Function GetNameOf$(DeviceName$, DeviceHandle&, API_ID&)
Dim NameBuffer As UNameType
'
' First need to get the length of the name string
Status& = DeviceIoControl(DeviceHandle&, API_ID&, 0, NameBuffer.Length, 260, BytesReturned&, 0)
If Status& = 0 Then ErrorExit ("Could not get LENGTH of " & DeviceName$ & " Name")
If NameBuffer.Length > 256 Then ErrorExit (Name$ & " Name > 256 Characters")
'
' . . . and then the string. It will be returned in UNICODE format

```

```

Status& = DeviceIoControl(DeviceHandle&, API_ID&, NameBuffer.Length, NameBuffer.Length, NameBuffer.Length, NameBuffer.Length,
BytesReturned&, 0)
If Status& = 0 Then ErrorExit ("Could not get TEXT of " & DeviceName$ & " Name")
temp$ = "": i = 0 'A simple unicode to basic string conversion
Do While NameBuffer.UnicodeName(i) <> 0: temp$ = temp$ & Chr(NameBuffer.UnicodeName(i)): i = i + 2: Loop
GetNameOf$ = temp$ 'StrConv(NameBuffer.Length, vbFromUnicode)
End Function

Function GetExternalHubName$(ConnectionIndex&, DeviceHandle&)
Dim NameBuffer As UNodeType
'
' First need to get the length of the name string
NameBuffer.ConnectionIndex = ConnectionIndex
Status& = DeviceIoControl(DeviceHandle&, &H220414, NameBuffer.ConnectionIndex, 260, NameBuffer.ConnectionIndex, 260, BytesReturned&,
CNull)
If Status& = 0 Then ErrorExit ("Could not get LENGTH of External Hub Name")
If NameBuffer.Length > 256 Then ErrorExit ("External Hub Name > 256 Characters")
'
' . . . and then the string. It will be returned in UNICODE format
NameBuffer.ConnectionIndex = ConnectionIndex
Status& = DeviceIoControl(DeviceHandle&, &H220414, NameBuffer.ConnectionIndex, NameBuffer.Length, NameBuffer.ConnectionIndex,
NameBuffer.Length, BytesReturned&, 0)
If Status& = 0 Then ErrorExit ("Could not get TEXT of External Hub Name")
temp$ = "": i = 0
Do While NameBuffer.UnicodeName(i) <> 0: temp$ = temp$ & Chr(NameBuffer.UnicodeName(i)): i = i + 2: Loop
GetExternalHubName$ = temp$
End Function

```