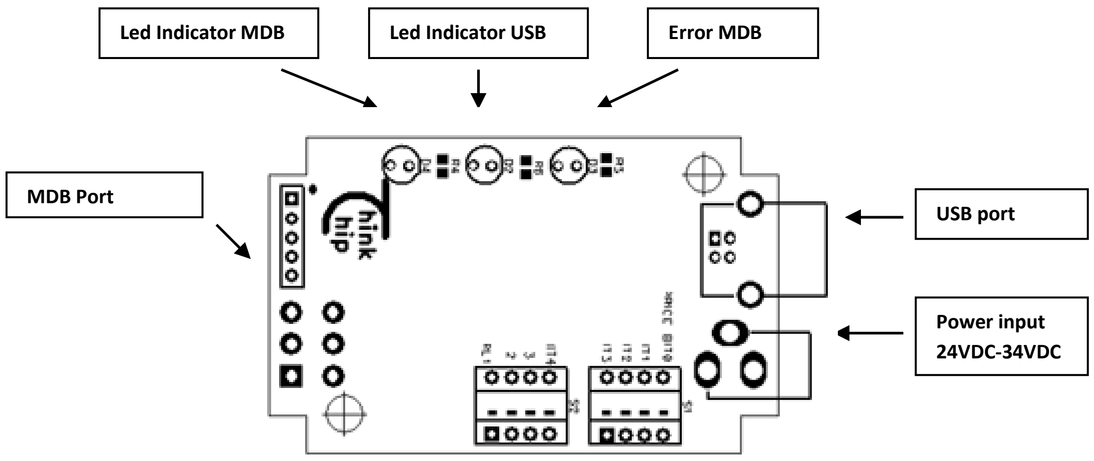
ThinkChip

Contenido

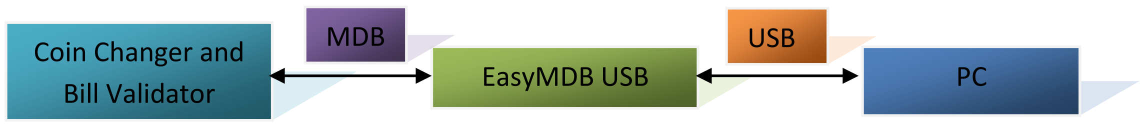
Introduction

In this guide, the user will find all technical information for the interface, and how to use the DLL file “SimpleUSB.dll” in the software Visual Studio.

Hardware



Power input: 24VDC-34VDC 2Amp.
USB: 5VDC 500 mA

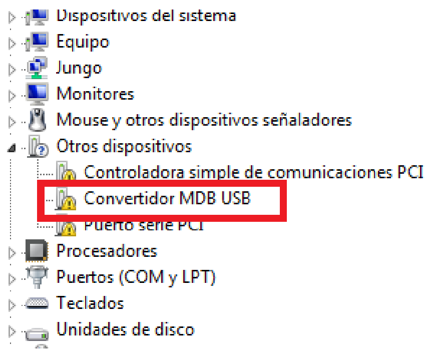


ThinkChip

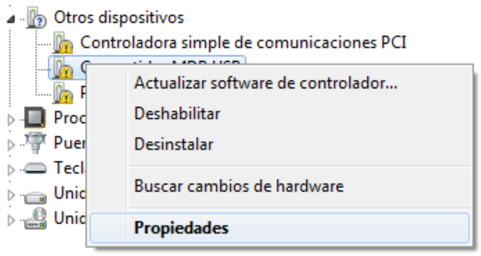
Install guide

When connecting the MDB USB Device, the Windows operating system will try to search their database driver corresponding, if there not finding, then will ask the user to manually install driver. For Windows XP is automatically request manually install the driver, but for Windows Vista, 7 and 8, install from Device Manager.

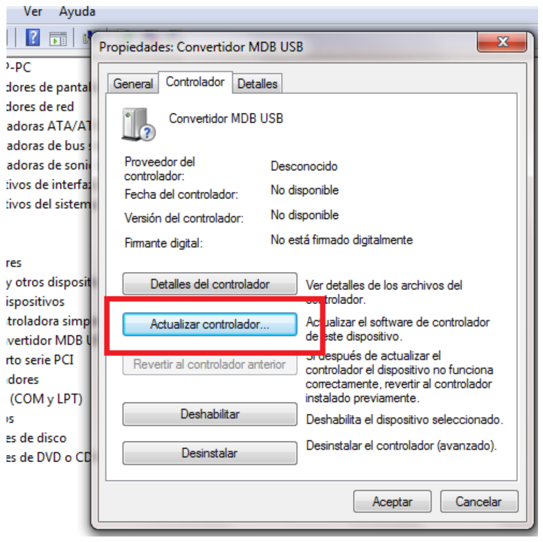
When the device is connected, will appear an icon in the Device Manager as seen in the figure.



Select the device right-click, and select Properties


Otros dispositivos
Controladora simple de comunicaciones PCI
Actualizar software de controlador...
Deshabilitar
Desinstalar
Buscar cambios de hardware
Propiedades

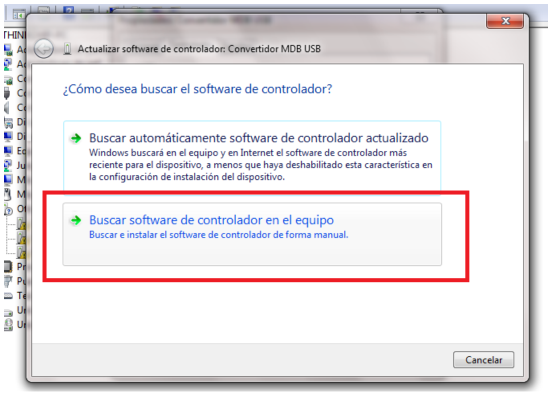
A window will appear, in which select Update Driver



Proveedor del controlador: Desconocido
Fecha del controlador: No disponible
Versión del controlador: No disponible
Firmante digital: No está firmado digitalmente

Ver detalles de los archivos del controlador.
Actualizar el software de controlador de este dispositivo.
Si después de actualizar el controlador el dispositivo no funciona correctamente, revertir al controlador instalado previamente.
Deshabilita el dispositivo seleccionado.
Desinstalar el controlador (avanzado).

Select "Search computer for driver software"



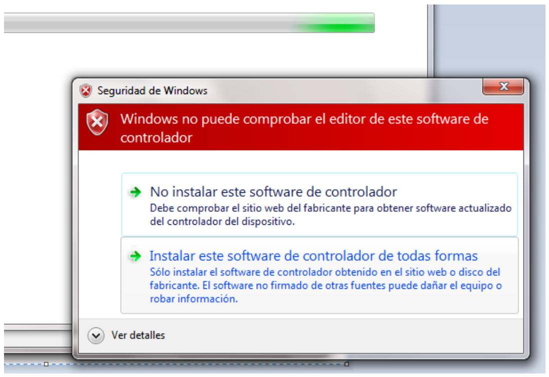
Actualizar software de controlador: Convertidor MDB USB

¿Cómo desea buscar el software de controlador?

Buscar automáticamente software de controlador actualizado
Windows buscará en el equipo y en Internet el software de controlador más reciente para el dispositivo, a menos que haya deshabilitado esta característica en la configuración de instalación del dispositivo.

Buscar software de controlador en el equipo
Buscar e instalar el software de controlador de forma manual.

We look for the folder where the driver called "USB MDB Converter" to accept and install, the following window appears, alerting is, this happens because Windows is not certified in the USB device, select "Install this driver software anyway "



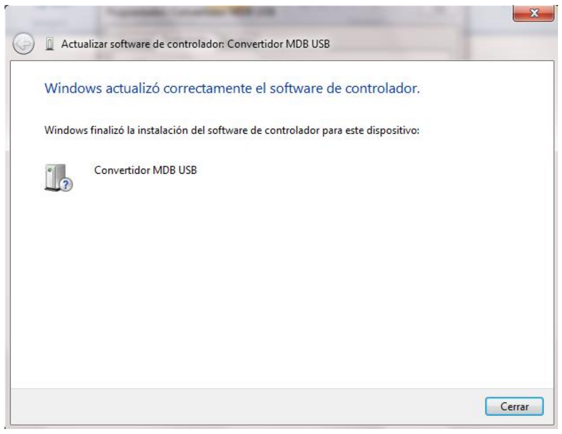
Seguridad de Windows

Windows no puede comprobar el editor de este software de controlador

No instalar este software de controlador
Debe comprobar el sitio web del fabricante para obtener software actualizado del controlador del dispositivo.

Instalar este software de controlador de todas formas
Sólo instalar el software de controlador obtenido en el sitio web o disco del fabricante. El software no firmado de otras fuentes puede dañar el equipo o robar información.

If you have successfully installed, the following window appears:



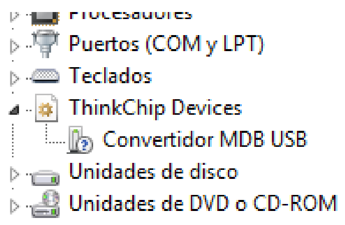
Actualizar software de controlador: Convertidor MDB USB

Windows actualizó correctamente el software de controlador.

Windows finalizó la instalación del software de controlador para este dispositivo:

Convertidor MDB USB

If you installed the driver successfully, be displayed in Device Manager

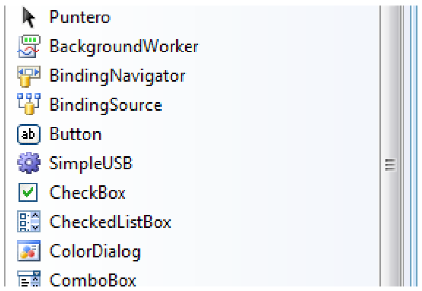


ThinkChip

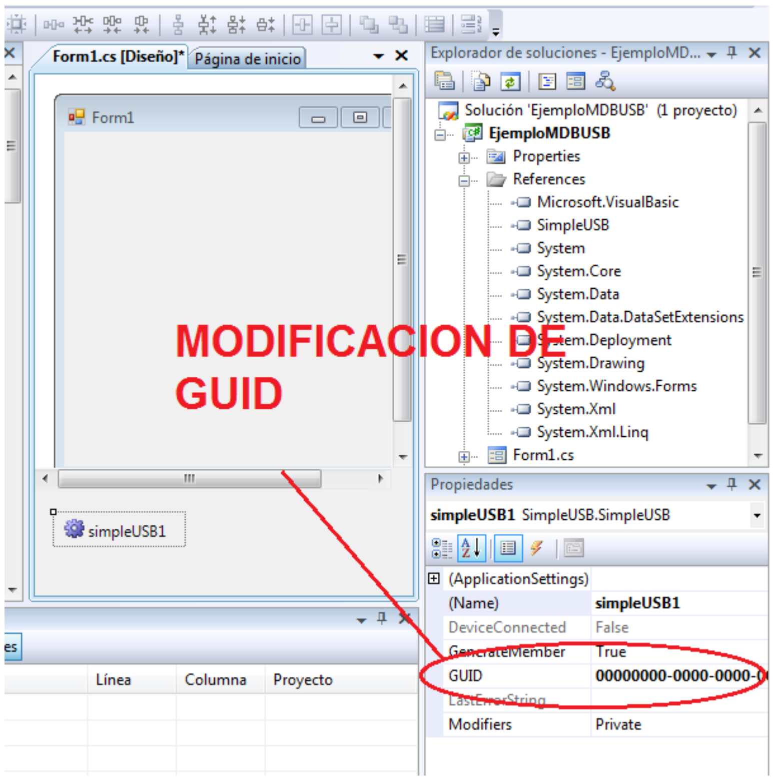
Software

Workspace in Visual Studio

There are two ways to integrate the DLL, we present the first way
Before starting to use the DLL, we need to add it to our tools, we need only search the "Software" folder in the direction where we copy the files on the disk, the file named "SimpleUSB.dll" and drag tools in Visual Studio.



When you select the new tool and integrate in to our Form, we set up a property called GUID.



(ApplicationSettings)	
(Name)	simpleUSB1
DeviceConnected	False
GenerateMember	True
GUID	00000000-0000-0000-
LastErrorString	
Modifiers	Private

ThinkChip

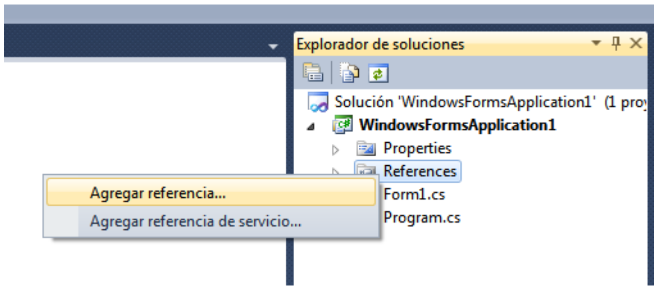
By default is 00000000-0000-0000-0000-000000000000, this number need to be changed by:

a3c4dc3e-683a-4220-9279-cdba089ea343

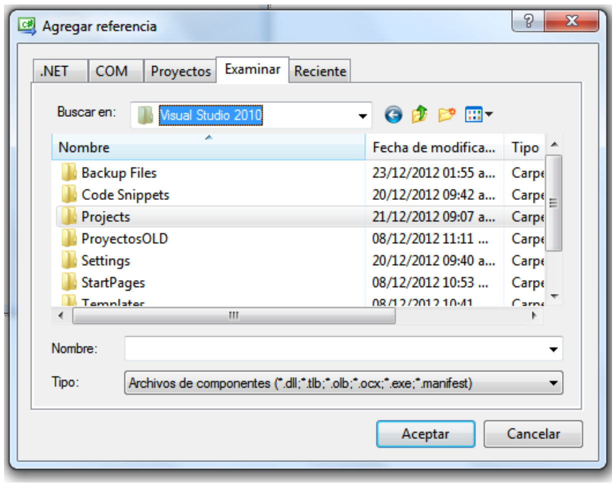
After changing this property, we can start working on the program. In the DLL there are different functions, then explain each, and an example of use. In the following examples, the "SimpleUSB" tool is left with the default name is "simpleUSB1".

Importing the DLL manually

Once the project is created we will import the dll "SimpleUSB" as follows: We will go to the right side to the "Solution Explorer" in the "References" tab give right click and will press "Add Reference".



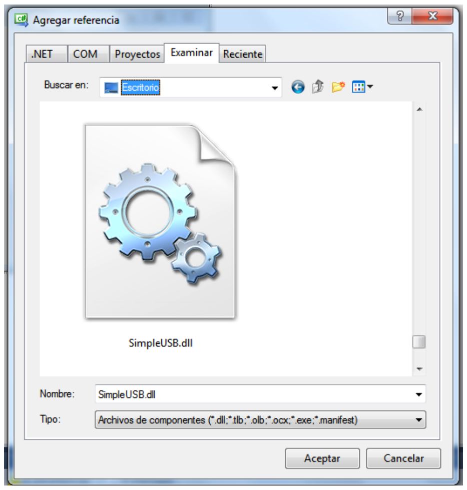
Open a window, you will click on the "Browse" tab and be where we will place the file "SimpleUSB.dll", select it and we will accept.



Nombre	Fecha de modifica...	Tipo
Backup Files	23/12/2012 01:55 a...	Carpe
Code Snippets	20/12/2012 09:42 a...	Carpe
Projects	21/12/2012 09:07 a...	Carpe
ProyectosOLD	08/12/2012 11:11 ...	Carpe
Settings	20/12/2012 09:40 a...	Carpe
StartPages	08/12/2012 10:53 ...	Carpe

Nombre:

Tipo: Archivos de componentes (*.dll;*.tlb;*.olb;*.ocx;*.exe;*.manifest)



Nombre: SimpleUSB.dll

Tipo: Archivos de componentes (*.dll;*.tlb;*.olb;*.ocx;*.exe;*.manifest)

Once completed these steps we will have added the dll to your project, but still we need settings the properties of DLL. In Visual C #, this is done using the keyword "using":

```
using SimpleUSB;
```

Done, just simply create an object of type SimpleUSB which allow us to use the functions or methods of the DLL:

```
namespace WindowsFormsApplication1
{
public partial class Form1 : Form
{
private SimpleUSB.SimpleUSB simpleUSB1 = new SimpleUSB.SimpleUSB();
```

Once the object is created and we can use this to configure some properties necessary for optimal working of our application with the USB device, such as the "GUID", the GUID is provided with the Interface MDB USB interface, this number always the same.

```
public Form1()
{
  InitializeComponent();
  simpleUSB1.GUID = new Guid("a3c4dc3e-683a-4220-9279-cdba089ea343");
```

It is advisable to do this from the "Load" event of the application.
To use the events generated by the USB, it will have to "connect" with the "functions" (or methods) where there will be the code to do when it happens [the event], this is done as follows:

```
this.simpleUSB1.onDeviceConnect += this.usb_onDeviceConnect;
this.simpleUSB1.onDeviceDisconnect += this.usb_onDeviceDisconnect;
this.simpleUSB1.onReadComplete += this.usb_onReadComplete;
```

Then these functions must be implemented:

```
    private void usb_onDeviceConnect()
{
            //Código a realizar
}

private void usb_onDeviceDisconnect()
{
            //Código a realizar
}


private void usb_onReadComplete()
```

```
{
            //Código a realizar
}
```

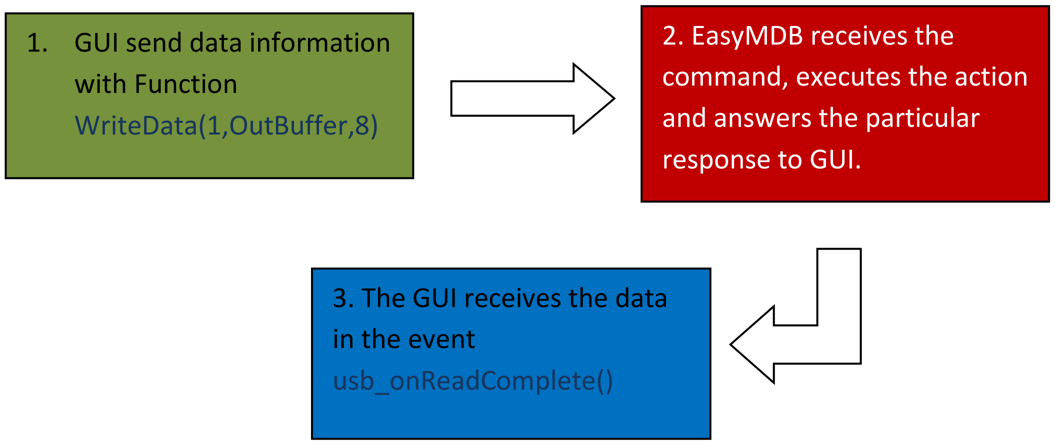
Type USB communication

For the EasyMDB USB communication, we handled two types of USB:

- Interrupt
- Bulk transfer

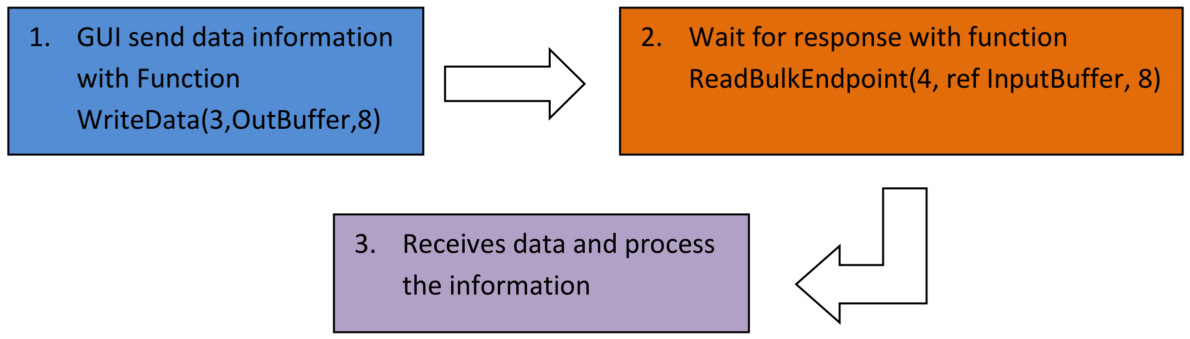
Interrupt mode

In the interrupt mode, and event is called every time that data is generated from Coin Changer and Bill Validator. The data could be generated automatically or by order to send data. The process that we receive the data is:


1. GUI send data information with Function WriteData(1,OutBuffer,8)
2. EasyMDB receives the command, executes the action and answers the particular response to GUI.
3. The GUI receives the data in the event usb_onReadComplete()

Bulk Transfer mode

For the Bulk Transfer mode, an event is not generated, so need wait for the answer, this mode is for function when need more data than 8 bytes.


1. GUI send data information with Function WriteData(3,OutBuffer,8)
2. Wait for response with function ReadBulkEndpoint(4, ref InputBuffer, 8)
3. Receives data and process the information

Connect and disconnect the device

To connect the device to the user interface, you need to first call the OpenConnection function ()
For the following example of using this function, you have created a button that when you call the event _Click the device is connected, this method can easily vary depending on your application.

```
private void button1_Click(object sender, EventArgs e)
    {
      simpleUSB1.OpenConnection();
    }
```

To disconnect the device you need to call the function CloseConnection ()
For the following example of using this function, you have created a button that when you call the event _Click the device is disconnected, this method can vary depending on your application easily.

```
private void button2_Click(object sender, EventArgs e)
    {
      simpleUSB1.CloseConnection();
    }
```

If you have successfully connected the MDB USB device, an event called _onDeviceConnect will invoke ()
If you successfully disconnected the device, called an event is invoked _onDeviceDisconnect ()
Example of use:

```
private void SimpleUSB1_onDeviceConnect()
    {
      label1.Text = "Connected";
      SimpleUSB1.StartReadingInterruptEndPoint(2, 8);
    }

private void SimpleUSB1_onDeviceDisconnect()
    {
      label1.Text = "Disconnected";
       simpleUSB1.StopReadingInterruptEndpoint();
    }
```

At any point in the program, we can ask if it is “ON” or “OFF” the device with the Boolean property DeviceConnected.

```
if (simpleUSB1.DeviceConnected = true)
        {
          //acción a realizar
        }
```

Start Reading data

To begin reading data from the device will be called the
StartReadingInterruptEndPoint function (2, 8) ;
You should call this function in the _onDeviceConnect () event, as seen in the example above, the parameters (2.8) are internal device performance data and should not be modified.

Stop Reading data

To stop reading data from the device, it will call the function
StopReadingInterruptEndpoint ()
You should call this function in the event _onDeviceDisconnect ()

Device Initialization

The device to be connected to the PC, start scanning if there MDB devices like Coin Changer and Bill Acceptors and, if not, see the Error LED flashes, this means that there is no device connected to the device to correctly

scan devices, must be of course at least one charging device is connected, and powered by a voltage source at least 2Amp 24VDC. If you have not turned on the power and 24VDC devices before connecting the MDB to PC USB converter, you need to initialize it again.
Before calling the function that will telling to MDB USB interface, which initialize the collection devices, you must create a buffer output data and input.

```
    byte[] OutputBuffer = new byte[32];
    byte[] InputBuffer = new byte[32];
```

Once created these buffer, will call the function WriteData (1, OutputBuffer, 8), where the first parameter "1" is not modified, the second parameter refers to the output buffer has been created, and the third parameter indicates the size of the data to send. In this example, a button will be pressed, calling the Click event, you ask before calling the function, the MDB USB drive when connected.

```
private void button3_Click(object sender, EventArgs e)
    {
      if (simpleUSB1.DeviceConnected == true)
      {
        OutputBuffer[0] = INIT_MDB_DEVICES; //ACTION TO BE REALIZED
        simpleUSB1.WriteData(1, OutputBuffer,8);
      }
    }
```

INIT_MDB_DEVICES is the constant that gives the order to initialize the cash devices, advancing as we go, we see the different constants supported by the interface MDB USB. Note that the constant INIT_MDB_DEVICES, is located in the 0 position of the string OutputBuffer.

As of now, most of the functions have an answer, the response function to initialize MDB recovery devices will focus on an event called:
_onReadComplete ()
By calling the event _onReadComplete, alerts the system that a new data comes MDB USB interface, but have not yet linked to where.
When the MDB USB device has finished to connect and initialized with Cash Devices (coin changer and bill acceptor) will respond in the event, in the following example, we asked at the location of buffer[0], if the response corresponds to the initialization, the location one of the string byte[1], specifies which devices are connected.

```
private void simpleUSB1_onReadComplete()
    {
      simpleUSB1.ReadInterruptEndpoint(ref InputBuffer, 8);

      if (InputBuffer[0] == INIT_MDB_DEVICES)
      {
        if (InputBuffer[1] == COIN_CHANGER)
        {
          label2.Text = "Coin Changer finded";
        }
        else if (InputBuffer[1] == BILL_ACCEPTOR)
        {
          label2.Text = "Bill Changer finded";
        }
        else if (InputBuffer[1] == COIN_AND_BILL)
        {
          label2.Text = "coin and bill connected";
        }
        else
        {
          label2.Text = "No cash devices";
        }
      }
    }
```

ThinkChip

Coin Changer Functions

Setup Coin Changer

In this function, we obtain the settings of coin changer, the parameters obtained are:

- Level: Indicates the feature level of the changer. This will distinguish the changers feature level to the VMC. This parameter is not important for use, only for data purposes.
- Coin Scaling factor: All accepted coin values must be evenly divisible by this number. For example, this could be set to 05H for the USA nickel.
- Decimal places: Indicates the number of decimal places on a credit display. For example, this could be set to 02H in the USA.
- Coin type routing: Indicates what coin types can be routed to the Changer's tubes.
- Country: The packed BCD country / currency code of the changer can be sent in two different forms depending on the value of the left most BCD digit. If the left most digit is a 0, the International Telephone Code is used to indicate the country that the changer is set-up for. For example, the USA code is 00 01H (Byte 1 = 00 and Byte 2 = 01). If the left most digit is a 1, the latest version of the ISO 4217 numeric currency code is used. For example, the code for the US dollar is 18 40H (Byte 1 = 18 and Byte 2 = 40) and for the Euro is 1978 (Byte 1 = 19 and Byte 2 = 78).

For obtain these data, we must first make the request data by function
WriteData (1,OutputBuffer, 8)

```
private void button4_Click(object sender, EventArgs e)
      {
         if (simpleUSB1.DeviceConnected == true)
         {
            OutputBuffer[0] = SETUP_COIN; //ACTION TO BE REALIZED
            simpleUSB1.WriteData(1, OutputBuffer,8);
         }
      }
```

The response will be sent to the event _onReadComplete ()
The following example reads the response from MDB USB drive, the data are sent as follows:

InputBuffer[0] = SETUP_COIN (CONSTANT)
InputBuffer[1] = Level
InputBuffer[2] = Scaling

InputBuffer[3] = Decimal Places
InputBuffer[4] = MSB of Coin Type Routing
InputBuffer[5] = LSB of Coin Type Routing
InputBuffer[6] = MSB of Country
InputBuffer[7] = LSB of Country

```
private void simpleUSB1_onReadComplete()
 {
        simpleUSB1.ReadInterruptEndpoint(ref InputBuffer, 8);

             byte msb, lsb;

            Level = InputBuffer[1];
            Scaling = InputBuffer[2];
            DecimalPlaces = InputBuffer[3];
            msb = InputBuffer[6];
            lsb = InputBuffer[7];
            Country = msb;
            Country <<= 8;
            Country |= lsb;

            textBox_status.Text += "Level Coin Changer: " + Level.ToString() + "\r\n";
            textBox_status.Text += "Scaling Coin Changer: " + Scaling.ToString() + "\r\n";
            textBox_status.Text += "Decimal places Changer: " + DecimalPlaces.ToString() + "\r\n";
            textBox_status.Text += "Country Coin Changer: " + Country.ToString() + "\r\n";
}
```

Cash Coin Inserted

When you have received money from the coin changer, the MDB USB drive automatically send the data through the event _onReadComplete ()

InputBuffer [0] = CASH_INSERTED (constant indicating that the data sent is money was inserted by user)
InputBuffer [1] = msb (most significant byte of the inserted amount)
InputBuffer [2] = lsb (least significant byte of the inserted amount)
InputBuffer [3] = routing (destination of the coin inserted)

```
Coin routing. 00: CASH BOX
              01: TUBES
              10: NOT USED
              11: REJECT
```

The following example shows how to receive the data and store it in a variable type int as well as print on a label for displaying the user.

```
private void simpleUSB1_onReadComplete()
 {
        simpleUSB1.ReadInterruptEndpoint(ref InputBuffer, 8);

         if (InputBuffer[0] == CASH_INSERTED)
         {
            byte msb, lsb;
            byte routing;
            int cash_inserted = 0;
            double cash;
            double cash_float;
            double scaling_float;

            msb = InputBuffer[1];
            lsb = InputBuffer[2];

            routing = InputBuffer[3];

            cash_inserted = msb;
            cash_inserted <<= 8;
            cash_inserted |= lsb;

            cash_float = Convert.ToDouble(cash_inserted);

               scaling_float = Convert.ToDouble(Scaling);
               cash = cash_float / scaling_float;

            textBox_status.Text += "Cash Inserted: $" + cash.ToString("#.##") + "\r\n";
            textBox_status.Text += "Routing: " + routing.ToString() + "\r\n";
         }
}
```

Error Data Coin Changer and Bill Acceptors

For errors cash devices, we have to call the function
WriteData (1, OutputBuffer, 8)

```
private void Button_Error_Click(object sender, EventArgs e)
    {
       OutputBuffer[0] = ERROR_STATUS; //ACTION TO BE REALIZED
       simpleUSB1.WriteData(1, OutputBuffer, 8);
    }
```

The information received in the _onReadComplete ()
Where in the InputBuffer receives data as follows:

InputBuffer [0] = constant that indicates the error status data
InputBuffer [1] = msb (most significant byte error coinchanger)
InputBuffer [2] = lsb (least significant byte error coinchanger)
InputBuffer [3] = ErrorBill (data error status bill acceptor)

The following example shows the reception of error, and prints a label status of each MDB devices.

```
 private void simpleUSB1_onReadComplete()
    {
       simpleUSB1.ReadInterruptEndpoint(ref InputBuffer, 8);
        if (InputBuffer[0] == ERROR_STATUS)
       {
          byte msb, lsb, error;
          int error_coinchanger;
          msb = InputBuffer[1];
          lsb = InputBuffer[2];
          error_coinchanger = msb;
          error_coinchanger <<= 8;
          error_coinchanger |= lsb;
          InputBuffer[3] = error;
          label4.Text = "Error coinchanger =" + error_coinchanger.ToString()
             + " Error bill=" + error.ToString();
       }
}
```

ThinkChip

Table Code Error for Coin Changer

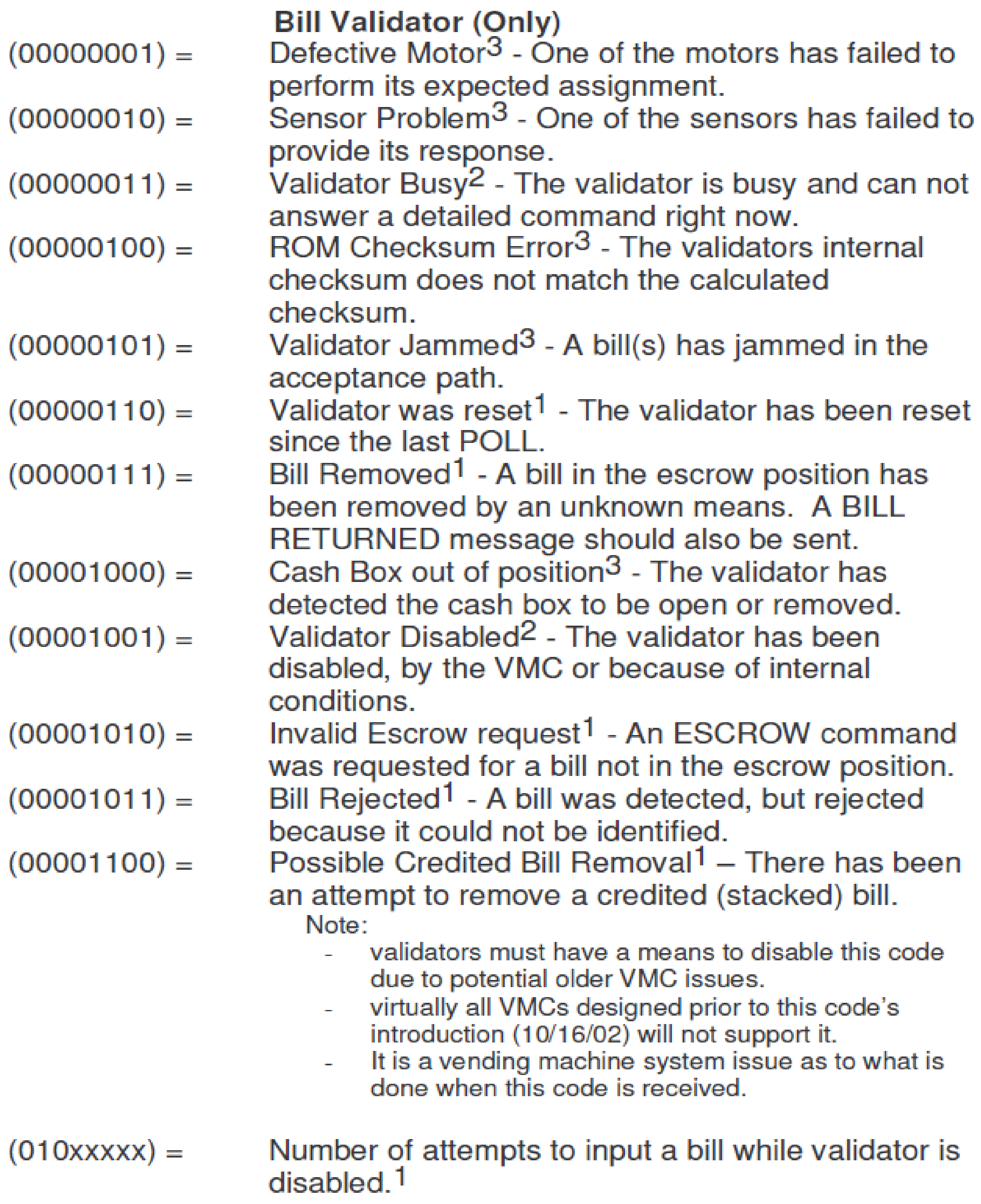
Z1 / Z2	Status	Cause(s) of Status / Error
01 / 00	Powering up	Changer powering up / initialization
02 / 00	Powering down	Changer powering down
03 / 00	OK	Changer fully operational and ready to accept coins
04 / 00	Keypad shifted	MODE key pressed and held so that LED flashes indicating keypad in shifted state. Reverts to normal mode if no key pressed for 15 seconds
05 / 10	Manual Fill / Payout active	Manual Fill or Manual Payout mode of operation in progress (under control of the changer). This response must be reported at least once to allow the VMC to request a manual fill or manual payout report.
05 / 20	New Inventory Information Available	Changer not in Manual inventory mode, but new inventory information available.
06 / 00	Inhibited by VMC	All coin acceptance inhibited at request of VMC, possibly due to product dispenser jams, completely sold out, etc.
10 / Z2	General changer error	Z2 defined as: **00** Non specific error. **01** Check sum error #1. A check sum error over a particular data range of configuration field detected. **02** Check sum error #2. A check sum error over a secondary data range or configuration field detected. **03** Low line voltage detected. The changer has disabled acceptance or payout due to a low voltage condition.

ThinkChip

Z1 / Z2	Status	Cause(s) of Status / Error
11 / Z2	Discriminator module error	Z2 defined as: **00** Non specific discriminator error. **10** Flight deck open. **11** Escrow Return stuck open. **30** Coin jam in sensor. **41** Discrimination below specified standard. **50** Validation sensor A out of range. The acceptor detects a problem with sensor A. **51** Validation sensor B out of range. The acceptor detects a problem with sensor B. **52** Validation sensor C out of range. The acceptor detects a problem with sensor C. **53** Operating temperature exceeded. The acceptor detects the ambient temperature has exceeded the changer's operating range, thus possibly affecting the acceptance rate. **54** Sizing optics failure. The acceptor detects an error in the sizing optics.
12 / Z2	Accept gate module error	Z2 defined as: **00** Non specific accept gate error. **30** Coins entered gate, but did not exit. **31** Accept gate alarm active. **40** Accept gate open, but no coin detected. **50** Post gate sensor covered before gate opened.
13 / Z2	Separator module error	Z2 defined as: **00** Non specific separator error **10** Sort sensor error. The acceptor detects an error in the sorting sensor.
14 / Z2	Dispenser module error	Z2 defined as: **00** Non specific dispenser error.
15 / Z2	Coin Cassette / tube module error	Z2 defined as: **00** Non specific cassette error. **02** Cassette removed. **03** Cash box sensor error. The changer detects an error in a cash box sensor. **04** Sunlight on tube sensors. The changer detects too much ambient light on one or more of the tube sensors.

ThinkChip

Table code error for bill Validator



Bill Validator (Only)

(00000001) =	Defective Motor[3] - One of the motors has failed to perform its expected assignment.
(00000010) =	Sensor Problem[3] - One of the sensors has failed to provide its response.
(00000011) =	Validator Busy[2] - The validator is busy and can not answer a detailed command right now.
(00000100) =	ROM Checksum Error[3] - The validators internal checksum does not match the calculated checksum.
(00000101) =	Validator Jammed[3] - A bill(s) has jammed in the acceptance path.
(00000110) =	Validator was reset[1] - The validator has been reset since the last POLL.
(00000111) =	Bill Removed[1] - A bill in the escrow position has been removed by an unknown means. A BILL RETURNED message should also be sent.
(00001000) =	Cash Box out of position[3] - The validator has detected the cash box to be open or removed.
(00001001) =	Validator Disabled[2] - The validator has been disabled, by the VMC or because of internal conditions.
(00001010) =	Invalid Escrow request[1] - An ESCROW command was requested for a bill not in the escrow position.
(00001011) =	Bill Rejected[1] - A bill was detected, but rejected because it could not be identified.
(00001100) =	Possible Credited Bill Removal[1] – There has been an attempt to remove a credited (stacked) bill. Note: - validators must have a means to disable this code due to potential older VMC issues. - virtually all VMCs designed prior to this code's introduction (10/16/02) will not support it. - It is a vending machine system issue as to what is done when this code is received.
(010xxxxx) =	Number of attempts to input a bill while validator is disabled.[1]

Coin Manually Dispensed

This event is generated automatically when a coin is dispensed manually, by pressing the buttons in the Coin Changer.

```
private void simpleUSB1_onReadComplete()
 {
        simpleUSB1.ReadInterruptEndpoint(ref InputBuffer, 8);

         if (InputBuffer[0] == CASH_DISPENSED_MANUALLY)
         {
            int valuecoins = 0;
            byte typecoin = 0;
            double realval = 0;
            double scaling_float;

            valuecoins = InputBuffer[1];
            valuecoins <<= 8;
            valuecoins |= InputBuffer[2];
            typecoin = InputBuffer[3];

            realval = Convert.ToDouble(valuecoins);
            scaling_float = Convert.ToDouble(Scaling);



            textBox_status.AppendText("Cash Dispensed Manually\r\n");
            textBox_status.AppendText("Amount:" + realval.ToString()+ "\r\n");
            textBox_status.AppendText("Type Coin" + typecoin.ToString() + "\r\n");



         }
}
```

Tubes Status Coin Changer

To know how much money is in the tubes in the coin changer, it will call the function

WriteData (1, BufferOutput,8)

In the following example, a button is used, calling the Click event

```
private void button4_Click(object sender, EventArgs e)
      {
         OutputBuffer[0] = CASH_IN_TUBES;
         simpleUSB1.WriteData(1, OutputBuffer, 8);
      }
```

The answer will be sent to the event _onReadComplete ()

The following example reads the response from MDB USB drive, the data are sent as follows:

InputBuffer [0] = constant indicating money tubes

InputBuffer [1] = msb (most significant data byte)

InputBuffer [2] = lsb (least significant data byte)

```
private void simpleUSB1_onReadComplete()
      {

        simpleUSB1.ReadInterruptEndpoint(ref InputBuffer, 8);
        if (InputBuffer[0] = CASH_IN_TUBES)
        {
           byte msb, lsb;
           int cash_in_tubes;

           msb = InputBuffer[1];
           lsb = InputBuffer[2];
           cash_in_tubes = msb;
           cash_in_tubes <<= 8;
           cash_in_tubes |= lsb;

           label5.Text = "Cash in tubes =$" + cash_in_tubes.ToString();
        }
}
```

ThinkChip

Send Cash Coin Changer (PayOut)

The amount is sent through a 16-bit integer, divided into 2 bytes, sending first the most significant and then the least significant output buffer in the order shown in more detail:

OutputBuffer [0] = constant that indicates the return cash
OutputBuffer [1] = msb (most significant byte)
OutputBuffer [2] = lsb (least significant byte)

```
private void Button_PayOut_Click(object sender, EventArgs e)
    {
        int cash_to_send;
        byte msb, lsb;
        decimal number;
        decimal scalingfloat;
        decimal numeric;

        scalingfloat= Convert.ToDecimal(Scaling);
        numeric = Numeric_PayOut.Value;

        number = numeric * scalingfloat;
        cash_to_send = Convert.ToInt16(number);

        msb = (byte)(cash_to_send >> 8);
        lsb = (byte)(cash_to_send & 0xff);
        OutputBuffer[0] = SEND_CHANGE;
        OutputBuffer[1] = msb;
        OutputBuffer[2] = lsb;

        if (simpleUSB1.DeviceConnected == true)
        {
          simpleUSB1.WriteData(1, OutputBuffer, 8);
        }

    }
```

Note that the variable Scaling was obtained from SETUP_COIN function
The USB converter MDB respond when finished the sending process of change, not necessarily be successful task and depends on the coin changer, later explained the possible answers. The following example shows how data is received from the _onReadComplete ()
Usually takes 1 to 5 seconds to finish to make change, depending on the model Coin Changer and the amount.

```
private void simpleUSB1_onReadComplete()
      {
        simpleUSB1.ReadInterruptEndpoint(ref InputBuffer, 8);
        if (InputBuffer[0] == SEND_CHANGE)
        {
            int valsendchanged = 0;
            double realval;
            double scaling_float;

            valsendchanged = InputBuffer[1];
            valsendchanged <<= 8;
            valsendchanged |= InputBuffer[2];
            //////////////part modified V1.1/////////////
            realval = Convert.ToDouble(valsendchanged);
            scaling_float = Convert.ToDouble(Scaling);



            textBox_status.AppendText("Send PayOut=" + realval.ToString() + "\r\n");

        }
}
```

The data sent to the InputBuffer[1] and InputBuffer[2] is the value amount that the Coin Changer dispense, for example, if send 10 value and received only 8, then miss 2 value for complete 10. Maybe because the Coin Changer don't have enough money in tubes or a fail in mechanism.

Type Coin Value Accepted

This function indicates the value of coin types 0 to 15. Values must be sent in ascending order. This number is the coin's monetary value divided by the coin scaling factor. Unused coin types are sent as 00H. Coin type credits sent as FFH are assumed to be vend tokens. That is, their value is assumed to worth one vend.
The bytes position in the 16 byte string indicates the coin type(s). For example, the first byte sent would indicate the value of coin type 0, the second byte sent would indicate the value of coin type 1, and so on. For example, the USA coin types may be; Coin type 0 = nickel, Coin type 1 = dime, Coin type 2 = quarter, Coin type 3 = dollar.

```
private void Button_coin_value_Click(object sender, EventArgs e)
    {
      OutputBuffer[0] = COIN_VALUE;
      OutputBuffer[1] = 1;
      simpleUSB1.WriteData(3, OutputBuffer, 8);
      simpleUSB1.ReadBulkEndpoint(4, ref InputBuffer, 8);

      if (InputBuffer[0] == COIN_VALUE && InputBuffer[1] == 1)
      {
        textBox_status.AppendText("Coin Value Extended Version\r\n");
        textBox_status.AppendText(InputBuffer[2].ToString() + "\r\n");
        textBox_status.AppendText(InputBuffer[3].ToString() + "\r\n");
        textBox_status.AppendText(InputBuffer[4].ToString() + "\r\n");
        textBox_status.AppendText(InputBuffer[5].ToString() + "\r\n");
        textBox_status.AppendText(InputBuffer[6].ToString() + "\r\n");
        textBox_status.AppendText(InputBuffer[7].ToString() + "\r\n");
      }
      OutputBuffer[0] = COIN_VALUE;
      OutputBuffer[1] = 2;
      simpleUSB1.WriteData(3, OutputBuffer, 8);
      simpleUSB1.ReadBulkEndpoint(4, ref InputBuffer, 8);
      if (InputBuffer[0] == COIN_VALUE && InputBuffer[1] == 2)
      {
        textBox_status.AppendText(InputBuffer[2].ToString() + "\r\n");
        textBox_status.AppendText(InputBuffer[3].ToString() + "\r\n");
        textBox_status.AppendText(InputBuffer[4].ToString() + "\r\n");
        textBox_status.AppendText(InputBuffer[5].ToString() + "\r\n");
        textBox_status.AppendText(InputBuffer[6].ToString() + "\r\n");
        textBox_status.AppendText(InputBuffer[7].ToString() + "\r\n");
      }

      OutputBuffer[0] = COIN_VALUE;
      OutputBuffer[1] = 3;
      simpleUSB1.WriteData(3, OutputBuffer, 8);
      simpleUSB1.ReadBulkEndpoint(4, ref InputBuffer, 8);

      if (InputBuffer[0] == COIN_VALUE && InputBuffer[1] == 3)
      {

        textBox_status.AppendText(InputBuffer[2].ToString() + "\r\n");
        textBox_status.AppendText(InputBuffer[3].ToString() + "\r\n");
        textBox_status.AppendText(InputBuffer[4].ToString() + "\r\n");
        textBox_status.AppendText(InputBuffer[5].ToString() + "\r\n");

      }
    }
```

Coin Accepted Extended Version

This function gives more options to the user, and could enable or disable each type of coin accepted.
Two values of 16 bits are the input parameters, the first data is for coin value accepted and the second is coin manually dispense enable.

Coin Enable

```
b15  b14  b13  b12  b11  b10  b9  b8 | b7  b6  b5  b4  b3  b2  b1  b0
Y1                                                                Y2
```

A bit is set to indicate a coin type is accepted. For example, bit 6 is set to indicate coin type 6, bit 15 is set to indicate coin type 15, and so on. To disable the changer, disable all coin types by sending a data block containing 0000H. All coins are automatically disabled upon reset.

Manually Coin Enable

```
b15  b14  b13  b12  b11  b10  b9  b8 | b7  b6  b5  b4  b3  b2  b1  b0
Y1                                                                Y2
```

A bit is set to indicate dispense enable. For example, bit 2 is set to enable dispensing of coin type 2. This command enables/disables manual dispensing using optional inventory switches. All manual dispensing switches are automatically enabled upon reset

```
int valaccepted=0x33;

 OutputBuffer[0] = SET_COINCHANGER_EXT;
  //coin enable
  OutputBuffer[1] = (byte)(valaccepted >> 8);//coin enable msb
  OutputBuffer[2] = (byte)(valaccepted & 0xff);//coin enable lsb
  //coin enable manually dispense, this value could be different that coin enable value
  OutputBuffer[3] = (byte)(valaccepted >> 8);//coin enable manually dispense msb
  OutputBuffer[4] = (byte)(valaccepted & 0xff);//coin enable manually dispense lsb
  simpleUSB1.WriteData(1, OutputBuffer, 8);
```

The response is received in the event usb_onReadComplete()

```
if (InputBuffer[0] == SET_COINCHANGER_EXT)
{
    textBox_status.AppendText("Set Coin Changer Extended Version\r\n");
    if (InputBuffer[1] == 1)
    {
        textBox_status.AppendText("OK\r\n");
    }
    else
    {
        textBox_status.AppendText("FAIL\r\n");
 }
```

Tubes Status Extended Version

This function is used to know how many coins are in each tube. Is the version extended of Tube Status, to receive the data we need to send the function 3 times, this function is Bulk Transfer data, and we need to receive 18 bytes.

WriteData (3,OutputBuffer, 8)

```
private void Button_TubeExt_Click(object sender, EventArgs e)
    {
      OutputBuffer[0] = CASH_IN_TUBES_EXT;
      OutputBuffer[1] = 1;
      simpleUSB1.WriteData(3, OutputBuffer, 8);
      simpleUSB1.ReadBulkEndpoint(4, ref InputBuffer, 8);
      if (InputBuffer[0] == CASH_IN_TUBES_EXT && InputBuffer[1]==1)
      {
        textBox_status.AppendText("Cash Tubes Extended Version\r\n");
        textBox_status.AppendText("Tube Full Status\r\n");
        textBox_status.AppendText(InputBuffer[2].ToString() + "\r\n");
        textBox_status.AppendText(InputBuffer[3].ToString() + "\r\n");
        textBox_status.AppendText("Cash Tubes Extended Version\r\n");
        textBox_status.AppendText("Tubes Status\r\n");
        textBox_status.AppendText(InputBuffer[4].ToString() + "\r\n");
        textBox_status.AppendText(InputBuffer[5].ToString() + "\r\n");
        textBox_status.AppendText(InputBuffer[6].ToString() + "\r\n");
        textBox_status.AppendText(InputBuffer[7].ToString() + "\r\n");
      }
      OutputBuffer[0] = CASH_IN_TUBES_EXT;
      OutputBuffer[1] = 2;
      simpleUSB1.WriteData(3, OutputBuffer, 8);
      simpleUSB1.ReadBulkEndpoint(4, ref InputBuffer, 8);
      if (InputBuffer[0] == CASH_IN_TUBES_EXT && InputBuffer[1] == 2)
      {
        textBox_status.AppendText(InputBuffer[2].ToString() + "\r\n");
        textBox_status.AppendText(InputBuffer[3].ToString() + "\r\n");
        textBox_status.AppendText(InputBuffer[4].ToString() + "\r\n");
        textBox_status.AppendText(InputBuffer[5].ToString() + "\r\n");
        textBox_status.AppendText(InputBuffer[6].ToString() + "\r\n");
        textBox_status.AppendText(InputBuffer[7].ToString() + "\r\n");
      }
      OutputBuffer[0] = CASH_IN_TUBES_EXT;
      OutputBuffer[1] = 3;
      simpleUSB1.WriteData(3, OutputBuffer, 8);
      simpleUSB1.ReadBulkEndpoint(4, ref InputBuffer, 8);
      if (InputBuffer[0] == CASH_IN_TUBES_EXT && InputBuffer[1] == 3)
      {
        textBox_status.AppendText(InputBuffer[2].ToString() + "\r\n");
        textBox_status.AppendText(InputBuffer[3].ToString() + "\r\n");
        textBox_status.AppendText(InputBuffer[4].ToString() + "\r\n");
        textBox_status.AppendText(InputBuffer[5].ToString() + "\r\n");
        textBox_status.AppendText(InputBuffer[6].ToString() + "\r\n");
        textBox_status.AppendText(InputBuffer[7].ToString() + "\r\n");
      }
    }
```

The first 2 bytes indicates status of coin tube for coin types 0 to 15.

```
b15  b14  b13  b12  b11  b10  b9  b8 | b7  b6  b5  b4  b3  b2  b1  b0
Y1                                                                  Y2
```

A bit is set to indicate a full tube. For example, bit 7 = set would indicate the tube for coin type 7 is full.

Indicates the greatest number of coins that the changer “knows” definitely are present in the coin tubes. A bytes position in the 16 byte string indicates the number of coins in a tube for a particular coin type. For example, the first byte sent indicates the number of coins in a tube for coin type 0. Unsent bytes are assumed to be zero. For tube counts greater than 255, counts should remain at 255.

Dispense Coin

This function is for dispense coin individually, don’t recommend to use for pay out, instead use PayOut function
To send data use the function WriteData (3,OutputBuffer, 8)
The parameters for this function is Types of Coin and number of coins.

```
private void Button_dispense_coin_Click(object sender, EventArgs e)
    {
        decimal numbercoin = 0;
        decimal typecoin = 0;

        numbercoin = numericUpDown_numbercoin.Value;
        typecoin = numericUpDown_typecoin.Value;


        OutputBuffer[0] = COIN_DISPENSE;
        OutputBuffer[1] = (byte)(numbercoin);
        OutputBuffer[2] = (byte)(typecoin);
        simpleUSB1.WriteData(1, OutputBuffer, 8);

    }
```

The response only tell that is done the action, but don’t tell us how much amount is given. The response is in the event usb_onReadComplete()

```
if (InputBuffer[0] == COIN_DISPENSE)
         {
            if (InputBuffer[1] == 1)
            {
               textBox_status.AppendText("Send Coin Dispense OK\r\n");
            }
            else
            {
               textBox_status.AppendText("Send Coin Dispense FAIL\r\n");
            }
         }
```

Bill Validator

Bills Inserted

When the Bill Validator receives a bill

The answer will be sent to the event _onReadComplete ()

The following example reads the response from MDB USB drive, the data are sent as follows:

InputBuffer [0] = constant indicating that this is Bill inserted

InputBuffer [1] = msb (most significant byte of the amount of bill)

InputBuffer [2] = lsb (least significant byte of the amount of bill)

InputBuffer [3] = Routing of bill (always be in escrow)

```
private void simpleUSB1_onReadComplete()
        {
          simpleUSB1.ReadInterruptEndpoint(ref InputBuffer, 8);
           if (InputBuffer[0] == BILL_INSERTED)
           {
               byte msb, lsb;
               byte routing;
               int cash_inserted = 0;
               double cash;
               double cash_float;
               double scaling_float;
               msb = InputBuffer[1];
               lsb = InputBuffer[2];
               routing = InputBuffer[3];
               cash_inserted = msb;
               cash_inserted <<= 8;
               cash_inserted |= lsb;
               cash_float = Convert.ToDouble(cash_inserted);
               // the scaling factor depends of the country value of bills that used
               //for example, in mexico use value of 20,50,100,200 and 500 MXN value
               //in the demo, use a bill acceptor brand Bellis technologies model BV20
               //not always is the same use of scaling value bill
               scaling_float = Convert.ToDouble(ScalingBill);
               cash = cash_float * 10.0;//only use for Mexican model BV20, should be change
                                  //for diferent country,model and brand company bill acceptor
               textBox_status.Text += "Cash Bills Inserted: $" + cash.ToString("#.##") + "\r\n";
               textBox_status.Text += "Routing Bill: " + routing.ToString() + "\r\n";
           }

        }

       }
```

Stacker Bill Validator

The inquiry Stacker bills gives us the number of bills in the escrow, not the amount of money. To request this information, the function is used:

WriteData (1,OutputBuffer, 8)

In the following example, information is requested by using the Click event of a button:

```
private void button5_Click(object sender, EventArgs e)
    {
      if (simpleUSB1.DeviceConnected == true)
      {
        OutputBuffer[0] = CASH_IN_BILL;
        simpleUSB1.WriteData(1, OutputBuffer, 8);
      }
    }
```

The response will be sent to the event _onReadComplete ()

Sending data is as follows in the InputBuffer

InputBuffer [0] = constant indicating that this is Stacker Bill

InputBuffer [1] = msb (most significant byte of the number of bills stack)

InputBuffer [2] = lsb (least significant byte of the number of bills stack)

The following example shows how to read the data, and printed on a label

```
private void simpleUSB1_onReadComplete()
      {
        simpleUSB1.ReadInterruptEndpoint(ref InputBuffer, 8);
        if (InputBuffer[0] = CASH_IN_BILL)
        {
           byte msb, lsb;
           int cash_in_bill = 0;

           msb = InputBuffer[1];
           lsb = InputBuffer[2];

           cash_in_bill = msb;
           cash_in_bill <<= 8;
           cash_in_bill |= lsb;

           label6.Text = "Numbers bills =" + cash_in_bill.ToString();

        }
}
```

Enable and disable Coin Changer and Bill Acceptor

In this function, we can enable or disable the MDB devices, depending on whether you want to enable or disable, is written to the output buffer, the order is as follows:

Enabling and Disabling coin changer
OutBuffer [0] = constant indicating enable coin changer
OutBuffer [1] = Boolean value indicating Enable = "1" disable = "0"

Enabling and Disabling bill validator
OutBuffer [0] = constant indicating enable bill validator
OutBuffer [1] = Boolean value indicating Enable = "1" disable = "0"

In the following example, four buttons for enabling and disabling the devices used:

```
 private void button6_Click(object sender, EventArgs e)
  {
    if (simpleUSB1.DeviceConnected == true)
    {
      OutputBuffer[0] = SET_COINCHANGER;
      OutputBuffer[1] = 1;
      simpleUSB1.WriteData(1,OutputBuffer,8);
    }
  }
  private void button7_Click(object sender, EventArgs e)
  {
    if (simpleUSB1.DeviceConnected == true)
    {
      OutputBuffer[0] = SET_COINCHANGER;
      OutputBuffer[1] = 0;
      simpleUSB1.WriteData(1, OutputBuffer, 8);
    }
  }
private void button9_Click(object sender, EventArgs e)
  {
    if (simpleUSB1.DeviceConnected == true)
    {
      OutputBuffer[0] = SET_BILL;
      OutputBuffer[1] = 1;
      simpleUSB1.WriteData(1, OutputBuffer, 8);
    }
  }
  private void button10_Click(object sender, EventArgs e)
  {
    if (simpleUSB1.DeviceConnected == true)
    {
      OutputBuffer[0] = SET_BILL;
      OutputBuffer[1] = 0;
      simpleUSB1.WriteData(1, OutputBuffer, 8);
    }
  }
```

Display MDB

This Function is for display a value in the MDB Display, available in the Store of the website www.thinkchip.info

The display has 4 digit to display a value 0 to 9999, in USA if you want to display a 1.50 value, a value of 150 need to be sent.

```
private void button1_Click(object sender, EventArgs e)
    {
      decimal value = 0;
      int cash = 0;
      byte msb, lsb;

      value = numericUpDown_display.Value;
      cash = Convert.ToInt16(value);

      msb = (byte)(cash >> 8);
      lsb = (byte)(cash & 0x00ff);


      OutputBuffer[0] = DISPLAY_VAL;
      OutputBuffer[1] = msb;
      OutputBuffer[2] = lsb;
      simpleUSB1.WriteData(3, OutputBuffer, 8);
      simpleUSB1.ReadBulkEndpoint(4, ref InputBuffer, 8);

      if (InputBuffer[0] == DISPLAY_VAL)
      {
        textBox_status.AppendText("Display MDB OK\r\n");
      }
    }
```

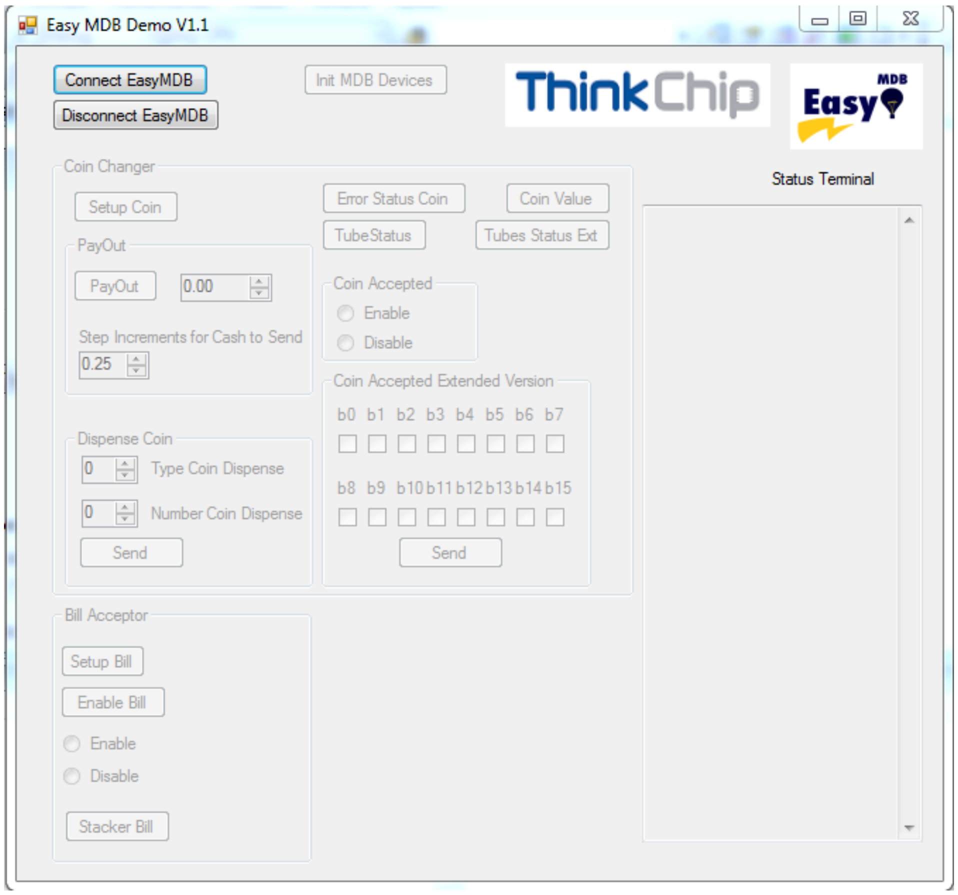
ThinkChip

CONSTANTS

```
 const byte INIT_MDB_DEVICES = 1;
const byte COIN_CHANGER = 1;
const byte BILL_ACCEPTOR = 2;
const byte COIN_AND_BILL = 3;
const byte CASH_INSERTED = 2;
const byte ERROR_STATUS = 4;
const byte CASH_IN_TUBES = 3;
const byte CASH_IN_TUBES_EXT = 13;
const byte CASH_IN_BILL = 6;
const byte SET_COINCHANGER = 10;
const byte SET_BILL = 11;
const byte SEND_CHANGE = 9;
const byte SETUP_COIN = 8;
const byte SETUP_BILL = 12;
const byte BILL_INSERTED = 5;
const byte COIN_VALUE = 14;
const byte SET_COINCHANGER_EXT = 15;
const byte CASH_DISPENSED_MANUALLY = 16;
const byte COIN_DISPENSE = 17;
```

Demo Code

The entire project is available in the demo software EasyMDB Example Demo, available in the web page www.thinkchip.info



Source Code

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using SimpleUSB;


/*
 *
 * <Example of how use the interface EasyMDB>
```

```
   Copyright (C) <2014>  <ThinkChip Braulio Chi Salavarria>

   This program is free software: you can redistribute it and/or modify
   it under the terms of the GNU General Public License as published by
   the Free Software Foundation, either version 3 of the License, or
   (at your option) any later version.

   This program is distributed in the hope that it will be useful,
   but WITHOUT ANY WARRANTY; without even the implied warranty of
   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
   GNU General Public License for more details.

   You should have received a copy of the GNU General Public License
   along with this program.  If not, see <http://www.gnu.org/licenses/>.
 *  */


namespace MDBUSBenglish
{
    public partial class Form1 : Form
    {

        byte[] OutputBuffer = new byte[32];
        byte[] InputBuffer = new byte[32];


        const byte INIT_MDB_DEVICES = 1;
        const byte COIN_CHANGER = 1;
        const byte BILL_ACCEPTOR = 2;
        const byte COIN_AND_BILL = 3;
        const byte CASH_INSERTED = 2;
        const byte ERROR_STATUS = 4;
        const byte CASH_IN_TUBES = 3;
        const byte CASH_IN_TUBES_EXT = 13;
        const byte CASH_IN_BILL = 6;
        const byte SET_COINCHANGER = 10;
        const byte SET_BILL = 11;
        const byte SEND_CHANGE = 9;
        const byte SETUP_COIN = 8;
        const byte SETUP_BILL = 12;
        const byte BILL_INSERTED = 5;
        const byte COIN_VALUE = 14;
        const byte SET_COINCHANGER_EXT = 15;
        const byte CASH_DISPENSED_MANUALLY = 16;
        const byte COIN_DISPENSE = 17;


        byte Level = 0;
        byte Scaling = 0;
        int Country = 0;
        byte DecimalPlaces = 0;

        int ScalingBill = 0;
        byte DecimalPlacesBill = 0;
        int CountryBill = 0;
        int StackerCapacity = 0;


        private SimpleUSB.SimpleUSB simpleUSB1 = new SimpleUSB.SimpleUSB();


        public Form1()
        {
           InitializeComponent();
           simpleUSB1.GUID = new Guid("a3c4dc3e-683a-4220-9279-cdba089ea343");
           groupBox_bill.Enabled = false;
           groupBox_coin.Enabled = false;
           Button_Init.Enabled = false;
           NumericSteps.Value = 0.25M;
           Numeric_PayOut.Increment = 0.25M;


        }

        private void Form1_Load(object sender, EventArgs e)
        {
           this.simpleUSB1.onDeviceConnect += this.usb_onDeviceConnect;
           this.simpleUSB1.onDeviceDisconnect += this.usb_onDeviceDisconnect;
           this.simpleUSB1.onReadComplete += this.usb_onReadComplete;


        }

        private void Button_connect_Click(object sender, EventArgs e)
        {
           simpleUSB1.OpenConnection();

        }


        private void usb_onDeviceConnect()
        {
           //
           textBox_status.AppendText("EasyMDB connected\r\n");
           simpleUSB1.StartReadingInterruptEndPoint(2, 8);
           Button_Init.Enabled = true;

        }

        private void usb_onDeviceDisconnect()
        {
           //
           textBox_status.AppendText("EasyMDB disconnected\r\n");
           simpleUSB1.StopReadingInterruptEndpoint();
           Button_Init.Enabled = false;
        }


        private void usb_onReadComplete()
        {
```

```
//


simpleUSB1.ReadInterruptEndpoint(ref InputBuffer, 8);

if (InputBuffer[0] == INIT_MDB_DEVICES)
{
   if (InputBuffer[1] == COIN_CHANGER)
   {
      textBox_status.AppendText("Coin Changer Connected\r\n");
      groupBox_coin.Enabled = true;

   }
   else if (InputBuffer[1] == BILL_ACCEPTOR)
   {
      textBox_status.AppendText("Bill Acceptor Connected\r\n");
      groupBox_bill.Enabled = true;
   }
   else if (InputBuffer[1] == COIN_AND_BILL)
   {
      textBox_status.AppendText("Coin and Bill Devices Connected\r\n");
      groupBox_bill.Enabled = true;
      groupBox_coin.Enabled = true;
   }
   else
   {
      textBox_status.AppendText("Not connected\r\n");
   }
}
if (InputBuffer[0] == SETUP_COIN)
{
   byte msb, lsb;

   Level = InputBuffer[1];
   Scaling = InputBuffer[2];
   DecimalPlaces = InputBuffer[3];
   msb = InputBuffer[6];
   lsb = InputBuffer[7];
   Country = msb;
   Country <<= 8;
   Country |= lsb;

   textBox_status.AppendText("Level Coin Changer: " + Level.ToString() + "\r\n");
   textBox_status.AppendText("Scaling Coin Changer: " + Scaling.ToString() + "\r\n");
   textBox_status.AppendText("Decimal places Changer: " + DecimalPlaces.ToString() + "\r\n");
   textBox_status.AppendText("Country Coin Changer: " + Country.ToString() + "\r\n");


}
if (InputBuffer[0] == CASH_INSERTED)
{
   byte msb, lsb;
   byte routing;
   int cash_inserted = 0;
   double cash;
```

```
    double cash_float;
    double scaling_float;

    msb = InputBuffer[1];
    lsb = InputBuffer[2];

    routing = InputBuffer[3];

    cash_inserted = msb;
    cash_inserted <<= 8;
    cash_inserted |= lsb;

    cash_float = Convert.ToDouble(cash_inserted);

        scaling_float = Convert.ToDouble(Scaling);
        cash = cash_float / scaling_float;

    textBox_status.AppendText("Cash Inserted: $" + cash.ToString("#.##") + "\r\n");
    textBox_status.AppendText("Routing: " + routing.ToString() + "\r\n");
}

if (InputBuffer[0] == ERROR_STATUS)
{
    byte msb, lsb, error;
    int error_coinchanger;

    msb = InputBuffer[1];
    lsb = InputBuffer[2];
    error_coinchanger = msb;
    error_coinchanger <<= 8;
    error_coinchanger |= lsb;

    error = InputBuffer[3];

    textBox_status.AppendText("Error coinchanger: " + error_coinchanger.ToString()
        + " Error bill: " + error.ToString() + "\r\n");


}
if (InputBuffer[0] == CASH_IN_TUBES)
{
    byte msb, lsb;
    int cash_in_tubes;
    double money;

    msb = InputBuffer[1];
    lsb = InputBuffer[2];
    cash_in_tubes = msb;
    cash_in_tubes <<= 8;
    cash_in_tubes |= lsb;

    money = cash_in_tubes/Scaling;
    textBox_status.AppendText("Cash in tubes :$" + money.ToString() + "\r\n");
}
```

```
if (InputBuffer[0] == CASH_IN_BILL)
{
   byte msb, lsb;
   int cash_in_bill = 0;

   msb = InputBuffer[1];
   lsb = InputBuffer[2];

   cash_in_bill = msb;
   cash_in_bill <<= 8;
   cash_in_bill |= lsb;

   textBox_status.AppendText("Number bills: " + cash_in_bill.ToString() + "\r\n");

}

if (InputBuffer[0] == SETUP_BILL)
{

   ScalingBill = InputBuffer[1];
   ScalingBill <<= 8;
   ScalingBill |= InputBuffer[2];
   DecimalPlacesBill = InputBuffer[3];
   StackerCapacity = InputBuffer[6];
   StackerCapacity <<= 8;
   StackerCapacity |= InputBuffer[7];
   CountryBill = InputBuffer[4];
   CountryBill <<= 8;
   CountryBill |= InputBuffer[5];



   textBox_status.AppendText("Scaling Bill Acceptor: " + ScalingBill.ToString() + "\r\n");
   textBox_status.AppendText("Stacker Capacity Bill Acceptor: " + StackerCapacity.ToString() + "\r\n");
   textBox_status.AppendText("Decimal places Bill: " + DecimalPlacesBill.ToString() + "\r\n");
   textBox_status.AppendText("Country Bill Acceptor: " + CountryBill.ToString() + "\r\n");

}

if (InputBuffer[0] == SEND_CHANGE)
{
   int valsendchanged = 0;
   double realval;
   double scaling_float;

   valsendchanged = InputBuffer[1];
   valsendchanged <<= 8;
   valsendchanged |= InputBuffer[2];
   /////////////part modified V1.1/////////////
   realval = Convert.ToDouble(valsendchanged);
   scaling_float = Convert.ToDouble(Scaling);



   textBox_status.AppendText("Send PayOut=" + realval.ToString() + "\r\n");
```

```
}
if (InputBuffer[0] == SET_BILL)
{
   textBox_status.AppendText("Set Bills OK\r\n");
}

if (InputBuffer[0] == BILL_INSERTED)
{
   byte msb, lsb;
   byte routing;
   int cash_inserted = 0;
   double cash;
   double cash_float;
   double scaling_float;

   msb = InputBuffer[1];
   lsb = InputBuffer[2];

   routing = InputBuffer[3];

   cash_inserted = msb;
   cash_inserted <<= 8;
   cash_inserted |= lsb;

   cash_float = Convert.ToDouble(cash_inserted);

   // the scaling factor depends of the country value of bills that used
   //for example, in mexico use value of 20,50,100,200 and 500 MXN value
   //in the demo, use a bill acceptor brand Bellis technologies model BV20
   //not always is the same use of scaling value bill
   scaling_float = Convert.ToDouble(ScalingBill);
   cash = cash_float * 10.0;//only use for Mexican model BV20, should be change
                     //for diferent country,model and brand company bill acceptor


   textBox_status.AppendText("Cash Bills Inserted: $" + cash.ToString("#.##") + "\r\n");
   textBox_status.AppendText("Routing Bill: " + routing.ToString() + "\r\n");
}

if (InputBuffer[0] == CASH_DISPENSED_MANUALLY)
{
   int valuecoins = 0;
   byte typecoin = 0;
   double realval = 0;
   double scaling_float;

   valuecoins = InputBuffer[1];
   valuecoins <<= 8;
   valuecoins |= InputBuffer[2];
   typecoin = InputBuffer[3];

   realval = Convert.ToDouble(valuecoins);
   scaling_float = Convert.ToDouble(Scaling);
```

```
            textBox_status.AppendText("Cash Dispensed Manually\r\n");
            textBox_status.AppendText("Amount:" + realval.ToString()+ "\r\n");
            textBox_status.AppendText("Type Coin" + typecoin.ToString() + "\r\n");




        }

        if (InputBuffer[0] == SET_COINCHANGER_EXT)
        {
            textBox_status.AppendText("Set Coin Changer Extended Version\r\n");
            if (InputBuffer[1] == 1)
            {
                textBox_status.AppendText("OK\r\n");
            }
            else
            {
                textBox_status.AppendText("FAIL\r\n");
            }

        }

        if (InputBuffer[0] == COIN_DISPENSE)
        {

            if (InputBuffer[1] == 1)
            {
                textBox_status.AppendText("Send Coin Dispense OK\r\n");
            }
            else
            {
                textBox_status.AppendText("Send Coin Dispense FAIL\r\n");
            }
        }






    }

    private void Button_disconnect_Click(object sender, EventArgs e)
    {
        simpleUSB1.CloseConnection();
    }

    private void Button_Init_Click(object sender, EventArgs e)
    {
        OutputBuffer[0] = INIT_MDB_DEVICES; //ACTION TO BE REALIZED
        simpleUSB1.WriteData(1, OutputBuffer, 8);
```

```
}

private void Button_SetupCoin_Click(object sender, EventArgs e)
{
   OutputBuffer[0] = SETUP_COIN; //ACTION TO BE REALIZED
   simpleUSB1.WriteData(1, OutputBuffer, 8);
}

private void Button_Tubes_Click(object sender, EventArgs e)
{
   OutputBuffer[0] = CASH_IN_TUBES; //ACTION TO BE REALIZED
   simpleUSB1.WriteData(1, OutputBuffer, 8);
}

private void Button_Error_Click(object sender, EventArgs e)
{
   OutputBuffer[0] = ERROR_STATUS; //ACTION TO BE REALIZED
   simpleUSB1.WriteData(1, OutputBuffer, 8);
}

private void Button_PayOut_Click(object sender, EventArgs e)
{
   int cash_to_send;
   byte msb, lsb;
   decimal number;
   decimal scalingfloat;
   decimal numeric;

   scalingfloat= Convert.ToDecimal(Scaling);
   numeric = Numeric_PayOut.Value;

   number = numeric * scalingfloat;
   cash_to_send = Convert.ToInt16(number);

   msb = (byte)(cash_to_send >> 8);
   lsb = (byte)(cash_to_send & 0xff);
   OutputBuffer[0] = SEND_CHANGE;
   OutputBuffer[1] = msb;
   OutputBuffer[2] = lsb;

   if (simpleUSB1.DeviceConnected == true)
   {
      simpleUSB1.WriteData(1, OutputBuffer, 8);
   }

}

private void Button_SetupBill_Click(object sender, EventArgs e)
{
   OutputBuffer[0] = SETUP_BILL; //ACTION TO BE REALIZED
   simpleUSB1.WriteData(1, OutputBuffer, 8);
}

private void textBox_status_TextChanged(object sender, EventArgs e)
{
}

private void NumericSteps_ValueChanged(object sender, EventArgs e)
{
   Numeric_PayOut.Increment = NumericSteps.Value;
}

private void Button_EnableBill_Click(object sender, EventArgs e)
{
   if (radioButton_enable.Checked == true)
   {
      OutputBuffer[0] = SET_BILL; //ACTION TO BE REALIZED
      OutputBuffer[1] = 1;
      simpleUSB1.WriteData(1, OutputBuffer, 8);

   }
   if (radioButton_disable.Checked == true)
   {
      OutputBuffer[0] = SET_BILL; //ACTION TO BE REALIZED
      OutputBuffer[1] = 0;
      simpleUSB1.WriteData(1, OutputBuffer, 8);

   }
}

private void Button_Stacker_Click(object sender, EventArgs e)
{
   OutputBuffer[0] = CASH_IN_BILL; //ACTION TO BE REALIZED
   simpleUSB1.WriteData(1, OutputBuffer, 8);

}

private void Button_TubeExt_Click(object sender, EventArgs e)
{
   OutputBuffer[0] = CASH_IN_TUBES_EXT;
   OutputBuffer[1] = 1;
   simpleUSB1.WriteData(3, OutputBuffer, 8);
   simpleUSB1.ReadBulkEndpoint(4, ref InputBuffer, 8);

   if (InputBuffer[0] == CASH_IN_TUBES_EXT && InputBuffer[1]==1)
   {
      textBox_status.AppendText("Cash Tubes Extended Version\r\n");
      textBox_status.AppendText("Tube Full Status\r\n");
      textBox_status.AppendText(InputBuffer[2].ToString() + "\r\n");
      textBox_status.AppendText(InputBuffer[3].ToString() + "\r\n");
      textBox_status.AppendText("Cash Tubes Extended Version\r\n");
      textBox_status.AppendText("Tubes Status\r\n");
      textBox_status.AppendText(InputBuffer[4].ToString() + "\r\n");
      textBox_status.AppendText(InputBuffer[5].ToString() + "\r\n");
      textBox_status.AppendText(InputBuffer[6].ToString() + "\r\n");
      textBox_status.AppendText(InputBuffer[7].ToString() + "\r\n");
   }

   OutputBuffer[0] = CASH_IN_TUBES_EXT;
```

```
            OutputBuffer[1] = 2;
            simpleUSB1.WriteData(3, OutputBuffer, 8);
            simpleUSB1.ReadBulkEndpoint(4, ref InputBuffer, 8);

            if (InputBuffer[0] == CASH_IN_TUBES_EXT && InputBuffer[1] == 2)
            {
                textBox_status.AppendText(InputBuffer[2].ToString() + "\r\n");
                textBox_status.AppendText(InputBuffer[3].ToString() + "\r\n");
                textBox_status.AppendText(InputBuffer[4].ToString() + "\r\n");
                textBox_status.AppendText(InputBuffer[5].ToString() + "\r\n");
                textBox_status.AppendText(InputBuffer[6].ToString() + "\r\n");
                textBox_status.AppendText(InputBuffer[7].ToString() + "\r\n");
            }

            OutputBuffer[0] = CASH_IN_TUBES_EXT;
            OutputBuffer[1] = 3;
            simpleUSB1.WriteData(3, OutputBuffer, 8);
            simpleUSB1.ReadBulkEndpoint(4, ref InputBuffer, 8);

            if (InputBuffer[0] == CASH_IN_TUBES_EXT && InputBuffer[1] == 3)
            {

                textBox_status.AppendText(InputBuffer[2].ToString() + "\r\n");
                textBox_status.AppendText(InputBuffer[3].ToString() + "\r\n");
                textBox_status.AppendText(InputBuffer[4].ToString() + "\r\n");
                textBox_status.AppendText(InputBuffer[5].ToString() + "\r\n");
                textBox_status.AppendText(InputBuffer[6].ToString() + "\r\n");
                textBox_status.AppendText(InputBuffer[7].ToString() + "\r\n");
            }

        }

        private void radioButton_enable_coin_CheckedChanged(object sender, EventArgs e)
        {
            if (radioButton_enable_coin.Checked == true)
            {
                OutputBuffer[0] = SET_COINCHANGER; //ACTION TO BE REALIZED
                OutputBuffer[1] = 1;
                simpleUSB1.WriteData(1, OutputBuffer, 8);
            }

            if (radioButton_enable_coin.Checked == false)
            {
                OutputBuffer[0] = SET_COINCHANGER;
                OutputBuffer[1] = 0;
                simpleUSB1.WriteData(1, OutputBuffer, 8);
            }

        }

        private void Button_coin_value_Click(object sender, EventArgs e)
        {
            OutputBuffer[0] = COIN_VALUE;
            OutputBuffer[1] = 1;
            simpleUSB1.WriteData(3, OutputBuffer, 8);
```

```
    simpleUSB1.ReadBulkEndpoint(4, ref InputBuffer, 8);

    if (InputBuffer[0] == COIN_VALUE && InputBuffer[1] == 1)
    {
        textBox_status.AppendText("Coin Value Extended Version\r\n");
        textBox_status.AppendText(InputBuffer[2].ToString() + "\r\n");
        textBox_status.AppendText(InputBuffer[3].ToString() + "\r\n");
        textBox_status.AppendText(InputBuffer[4].ToString() + "\r\n");
        textBox_status.AppendText(InputBuffer[5].ToString() + "\r\n");
        textBox_status.AppendText(InputBuffer[6].ToString() + "\r\n");
        textBox_status.AppendText(InputBuffer[7].ToString() + "\r\n");
    }

    OutputBuffer[0] = COIN_VALUE;
    OutputBuffer[1] = 2;
    simpleUSB1.WriteData(3, OutputBuffer, 8);
    simpleUSB1.ReadBulkEndpoint(4, ref InputBuffer, 8);

    if (InputBuffer[0] == COIN_VALUE && InputBuffer[1] == 2)
    {
        textBox_status.AppendText(InputBuffer[2].ToString() + "\r\n");
        textBox_status.AppendText(InputBuffer[3].ToString() + "\r\n");
        textBox_status.AppendText(InputBuffer[4].ToString() + "\r\n");
        textBox_status.AppendText(InputBuffer[5].ToString() + "\r\n");
        textBox_status.AppendText(InputBuffer[6].ToString() + "\r\n");
        textBox_status.AppendText(InputBuffer[7].ToString() + "\r\n");
    }

    OutputBuffer[0] = COIN_VALUE;
    OutputBuffer[1] = 3;
    simpleUSB1.WriteData(3, OutputBuffer, 8);
    simpleUSB1.ReadBulkEndpoint(4, ref InputBuffer, 8);

    if (InputBuffer[0] == COIN_VALUE && InputBuffer[1] == 3)
    {

        textBox_status.AppendText(InputBuffer[2].ToString() + "\r\n");
        textBox_status.AppendText(InputBuffer[3].ToString() + "\r\n");
        textBox_status.AppendText(InputBuffer[4].ToString() + "\r\n");
        textBox_status.AppendText(InputBuffer[5].ToString() + "\r\n");

    }
}

private void Button_coin_accepted_ext_Click(object sender, EventArgs e)
{
    int valaccepted = 0;

    if (checkBox_b0.Checked == true)
    {
        valaccepted = 0x01;
    }
    if (checkBox_b1.Checked == true)
    {
        valaccepted |= 0x02;
```

```
}
if (checkBox_b2.Checked == true)
{
   valaccepted |= 0x04;
}
if (checkBox_b3.Checked == true)
{
   valaccepted |= 0x08;
}
if (checkBox_b4.Checked == true)
{
   valaccepted |= 0x10;
}
if (checkBox_b5.Checked == true)
{
   valaccepted |= 0x20;
}
if (checkBox_b6.Checked == true)
{
   valaccepted |= 0x40;
}
if (checkBox_b7.Checked == true)
{
   valaccepted |= 0x80;
}
if (checkBox_b8.Checked == true)
{
   valaccepted |= 0x0100;
}
if (checkBox_b9.Checked == true)
{
   valaccepted |= 0x0200;
}
if (checkBox_b10.Checked == true)
{
   valaccepted |= 0x0400;
}
if (checkBox_b11.Checked == true)
{
   valaccepted |= 0x0800;
}
if (checkBox_b12.Checked == true)
{
   valaccepted |= 0x1000;
}
if (checkBox_b13.Checked == true)
{
   valaccepted |= 0x2000;
}
if (checkBox_b14.Checked == true)
{
   valaccepted |= 0x4000;
}
if (checkBox_b15.Checked == true)
{
```

```
            valaccepted  |= 0x8000;
          }

          OutputBuffer[0] = SET_COINCHANGER_EXT;
          //coin enable
          OutputBuffer[1] = (byte)(valaccepted >> 8);//coin enable msb
          OutputBuffer[2] = (byte)(valaccepted & 0xff);//coin enable lsb
          //coin enable manually dispense, this value could be different that coin enable value
          OutputBuffer[3] = (byte)(valaccepted >> 8);//coin enable manually dispense msb
          OutputBuffer[4] = (byte)(valaccepted & 0xff);//coin enable manually dispense lsb
          simpleUSB1.WriteData(1, OutputBuffer, 8);

        }

        private void Button_dispense_coin_Click(object sender, EventArgs e)
        {
          decimal numbercoin = 0;
          decimal typecoin = 0;

          numbercoin = numericUpDown_numbercoin.Value;
          typecoin = numericUpDown_typecoin.Value;


          OutputBuffer[0] = COIN_DISPENSE;
          OutputBuffer[1] = (byte)(numbercoin);
          OutputBuffer[2] = (byte)(typecoin);
          simpleUSB1.WriteData(1, OutputBuffer, 8);

        }




    }
}
```