CAN Bus .Net Framework Component

User Manual

Table of Contents

1 Features Highlights

- Support as many as 8 CAN bus Transmit Watchdogs, as many as 8 CAN Bus Receiving Watchdogs. The timer is based on hardware, not based on PC software.
- Support CAN BUS 2.0A and 2.0B.
- Support Remote Frame transmitting and receiving.
- Easily Program in VB.net/ C# /C++ program language under .net environment
- CAN Bus baud rate is some values from 25K to 1M bps
- 3 KV Isolation between CAN Bus and PC
- Power by external supply range is 5.5V to 28VDC or/and USB Connection.

2 Setup CAN Bus Component in Visual Studio

1. Firstly , Download CAN BUS .net component from our website: , unzip to any folder you like.
2. We take Visual Studio Community 2022 as an example. Run Visual Studio, and create one project as shown as below Fig.1: (You can use C#, but we take VB.net as an example)

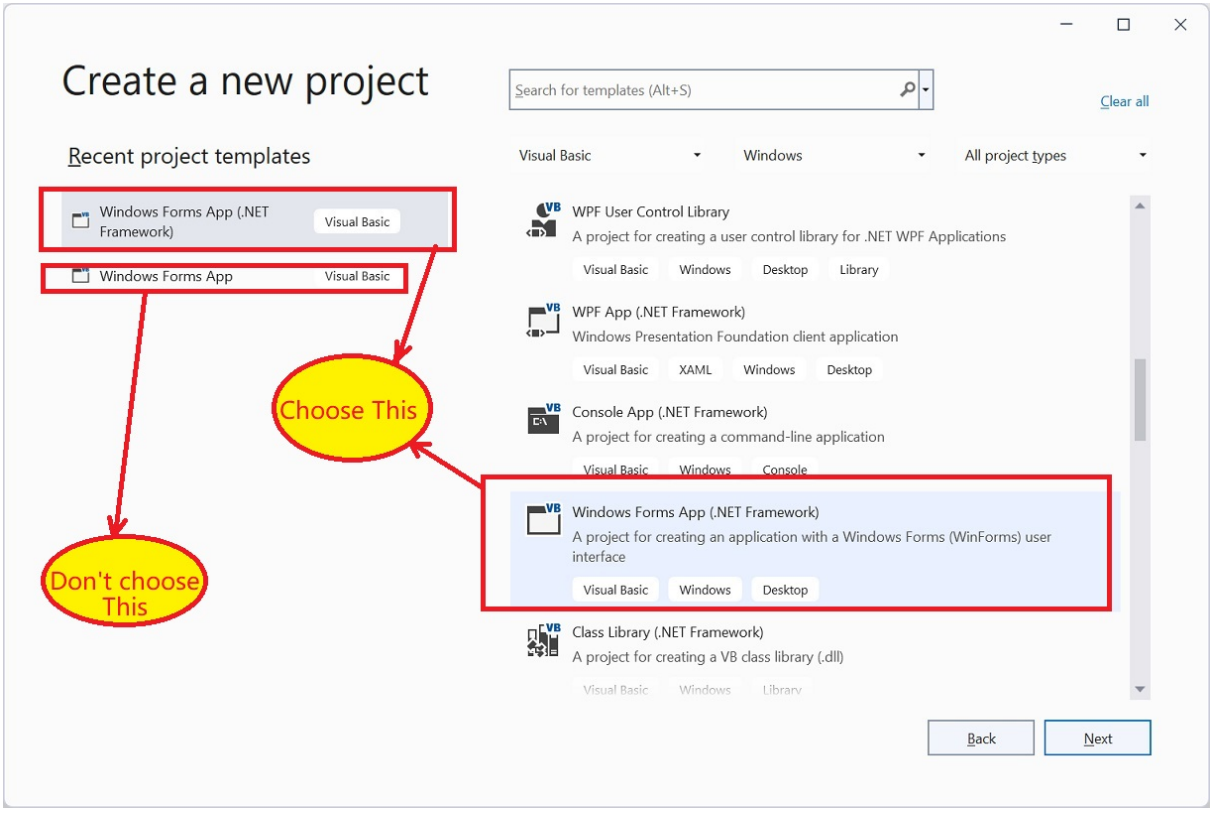

Fig.1 Create Windows Form App (.Net Framework)

3. Choose item as picture above , Click Next button , type Project name as Fig.2

.

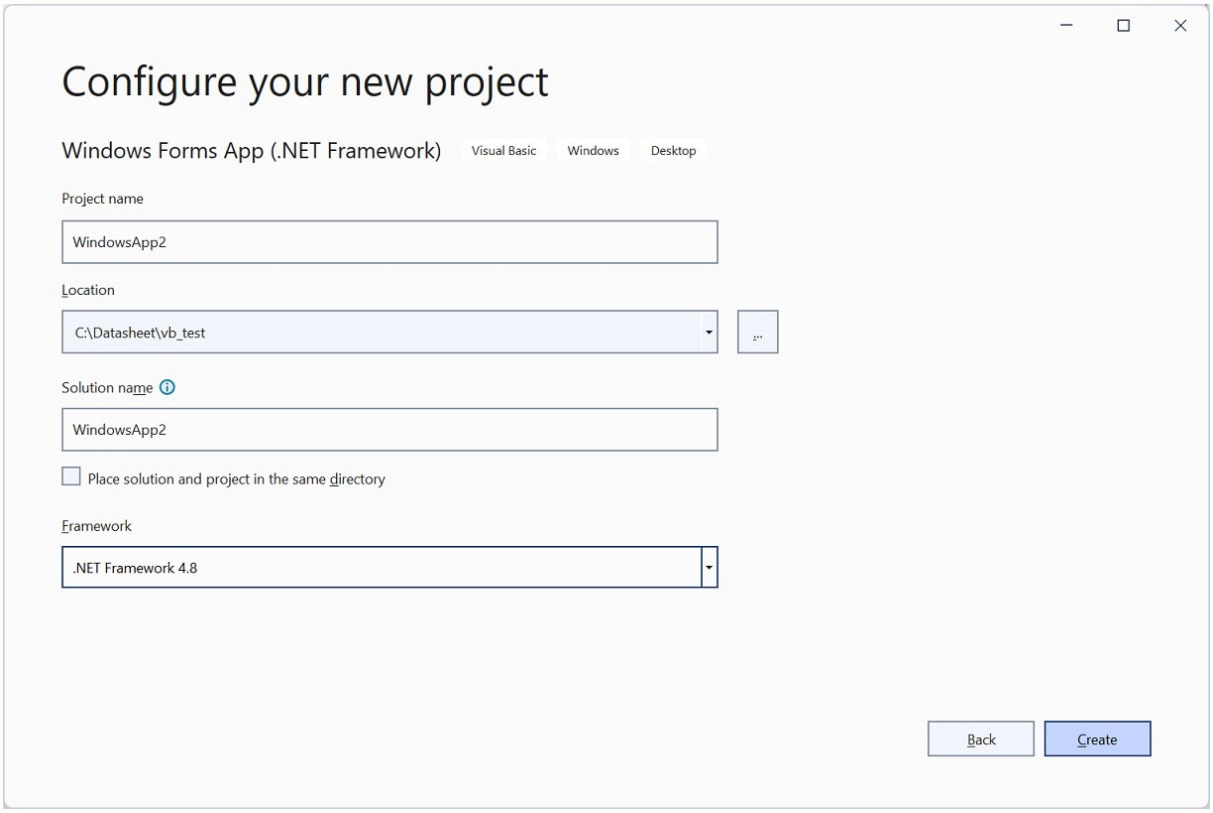

Fig.2 Project create

4. Click Create button in above picture Fig2, you will see Fig.3 below. In the Toolbox, righ click to pop up a menu, click "Add Tab"

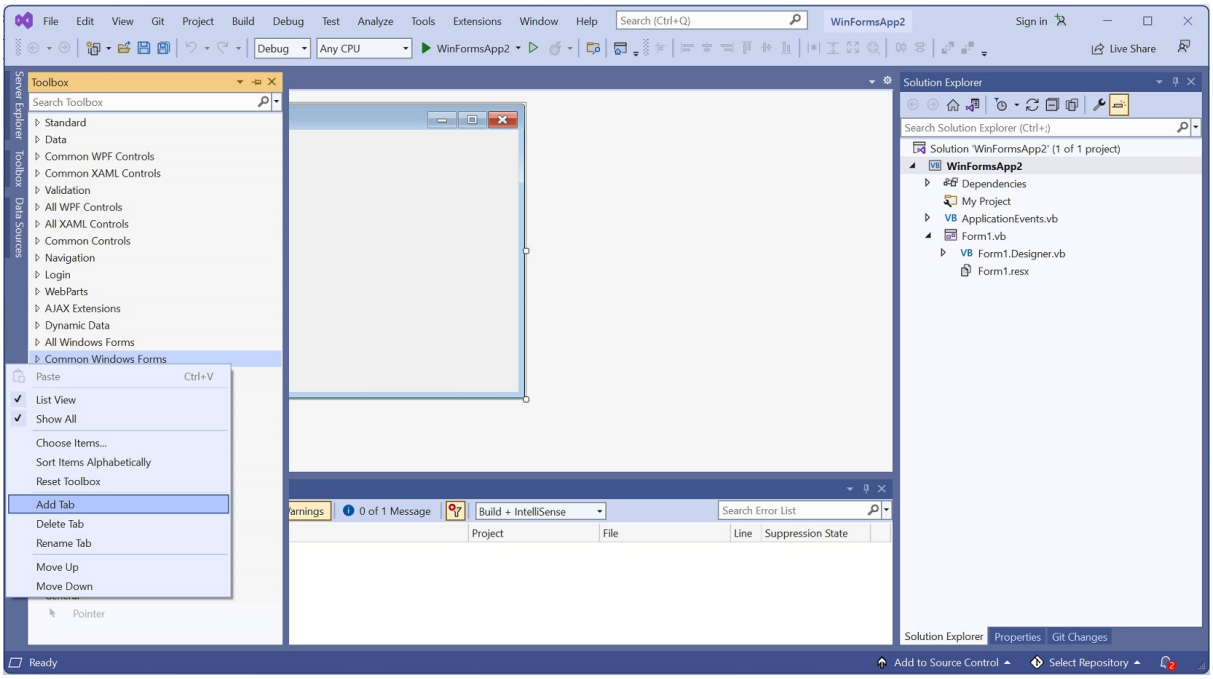

Fig. 3 Add Tab

5. Type any name you like, we type "Dafulai Electronics". We just created "Dafulai Electronics" tab as Fig.4 below:

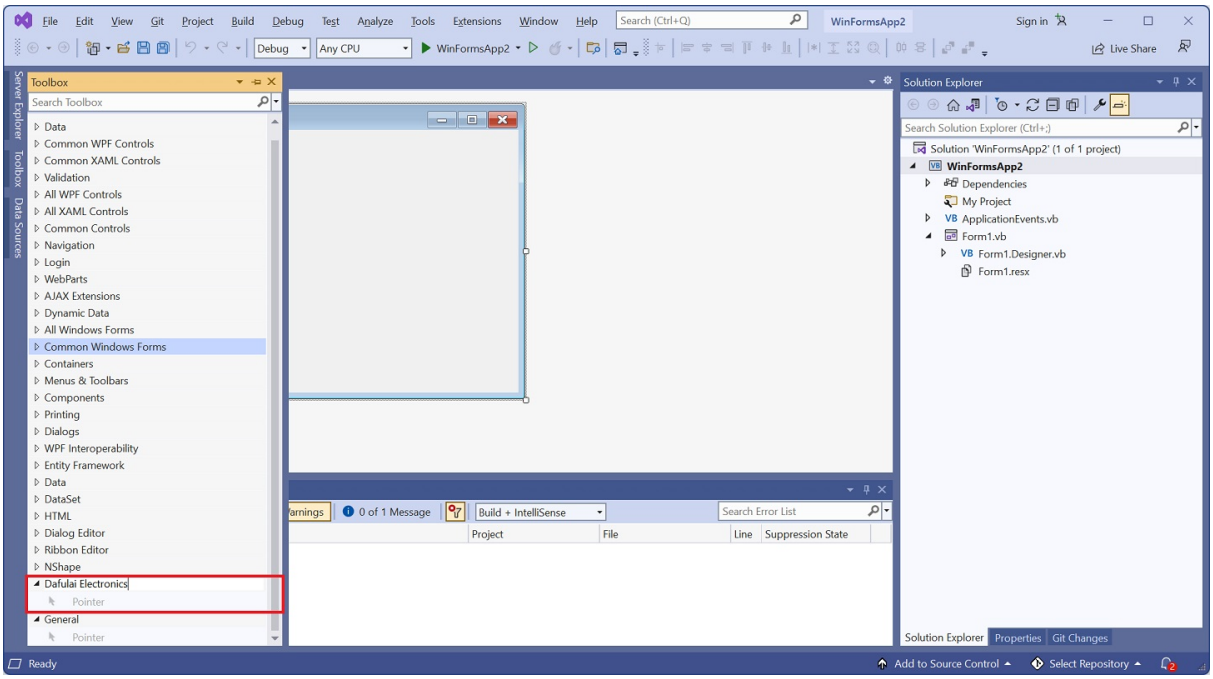

Fig.4 New Tab created

6. In above picture Fig.4, Right click on new created tab "Dafulai Electronics" to pop up menu as Fig.5 below:

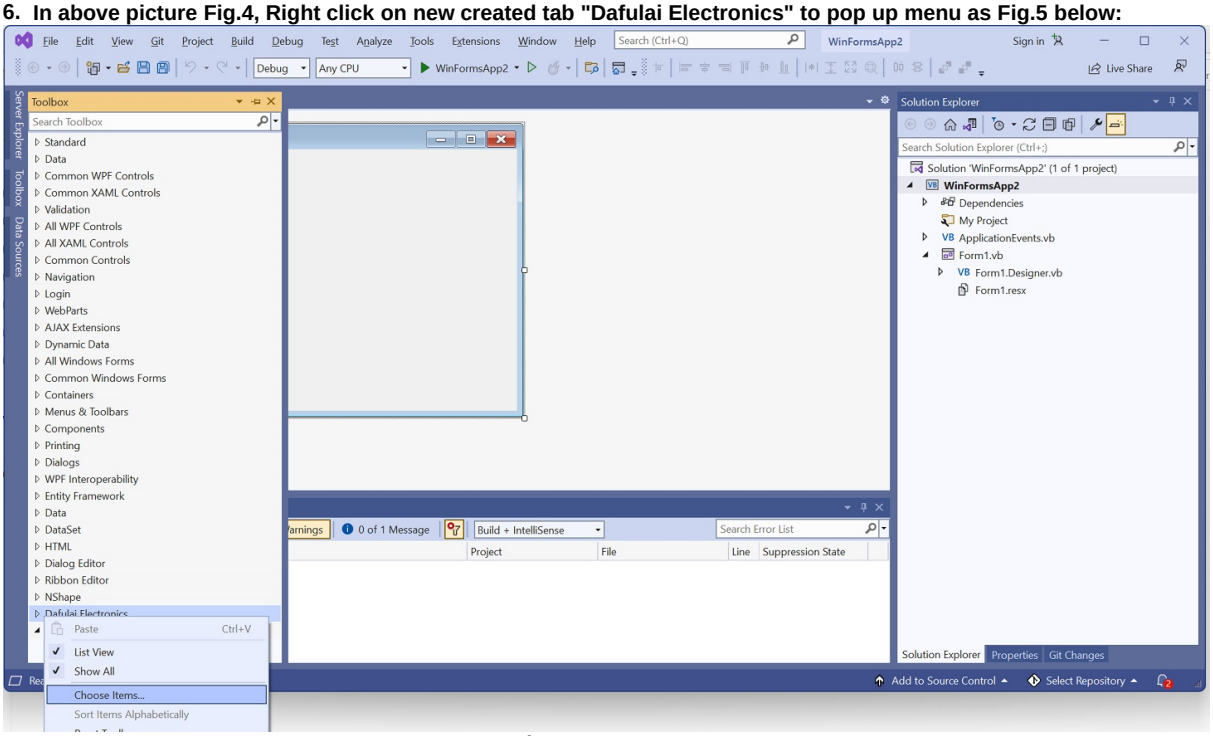

Fig 5 Popup Menu

7. In Fig.5 popup menu, click "Choose Items...". The following dialog Fig.6 will occur

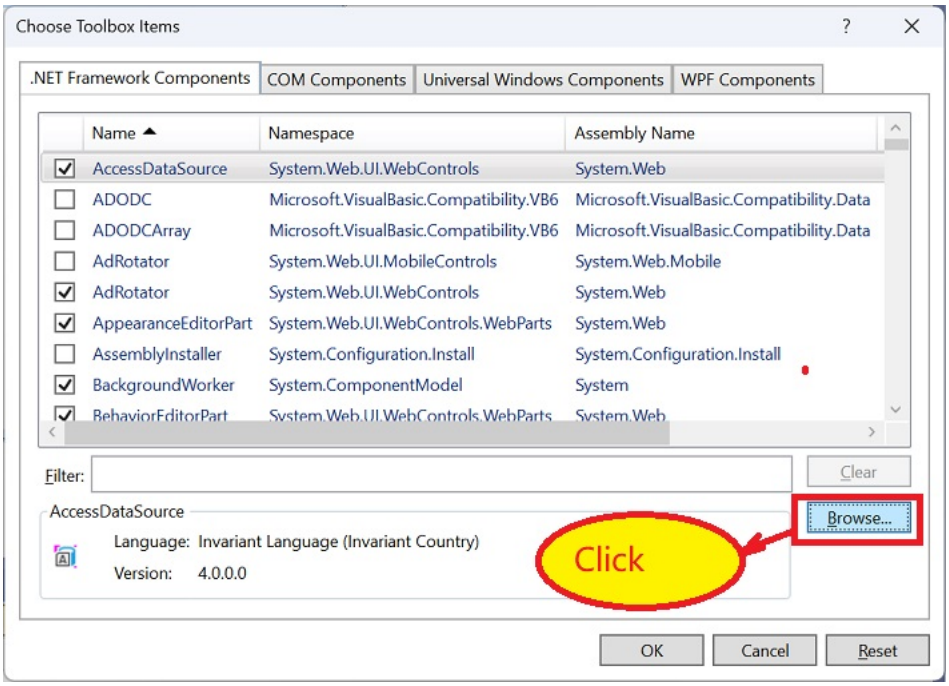

Fig. 6 Browe

8. Under tab ".Net Framewok Components", click "Browse..." button

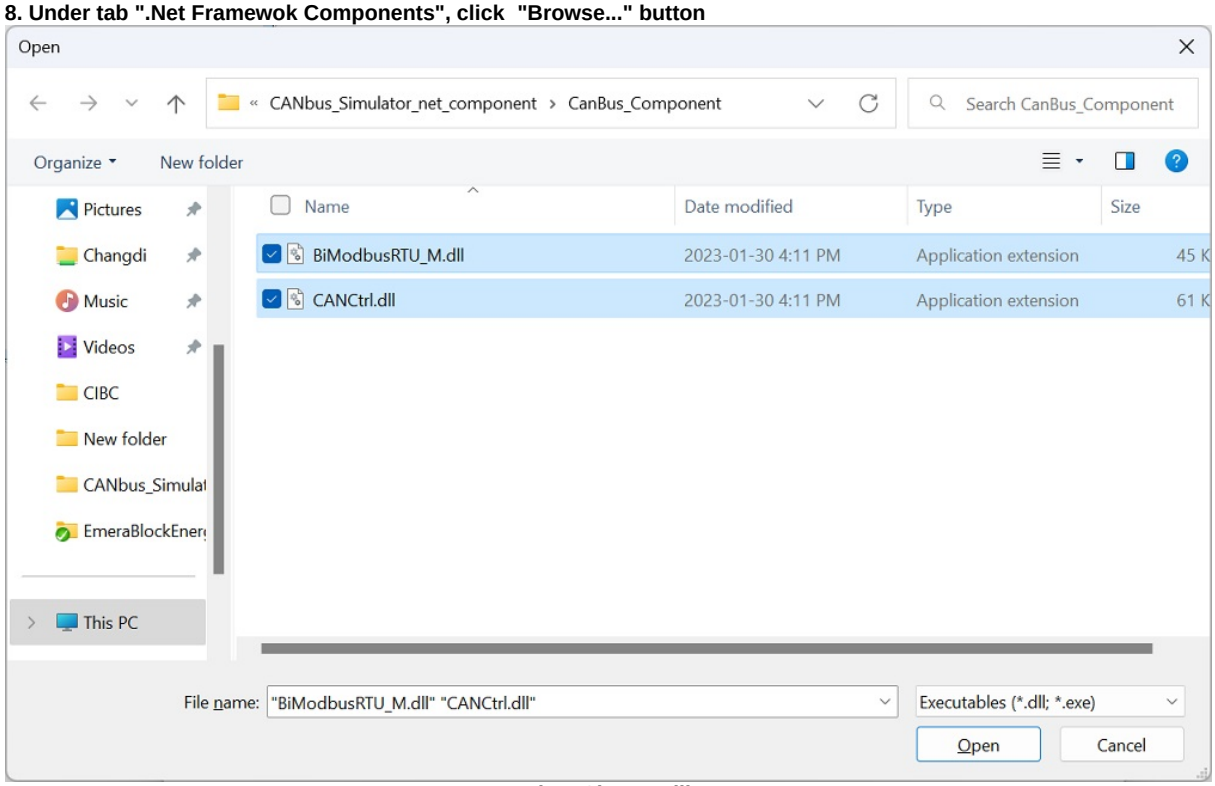

Fig.7 Choose dll

9. In Fig.7 above, please choose folder in step1 which is unzip files folder, and choose 2 Files: "CANCtrl.dll" and "BiModbusRTU_M.dll", and click "Open" button.

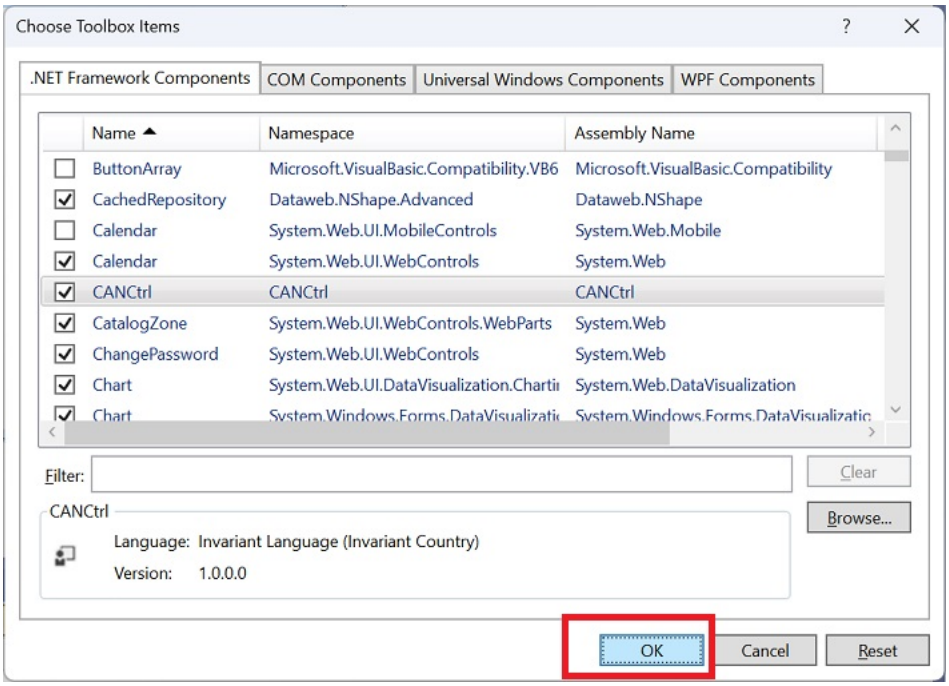

Fig.8 dll ok

10. Click "OK" in Fig. 8 above. You will see CANCtrl component occurred in Fig.9 below

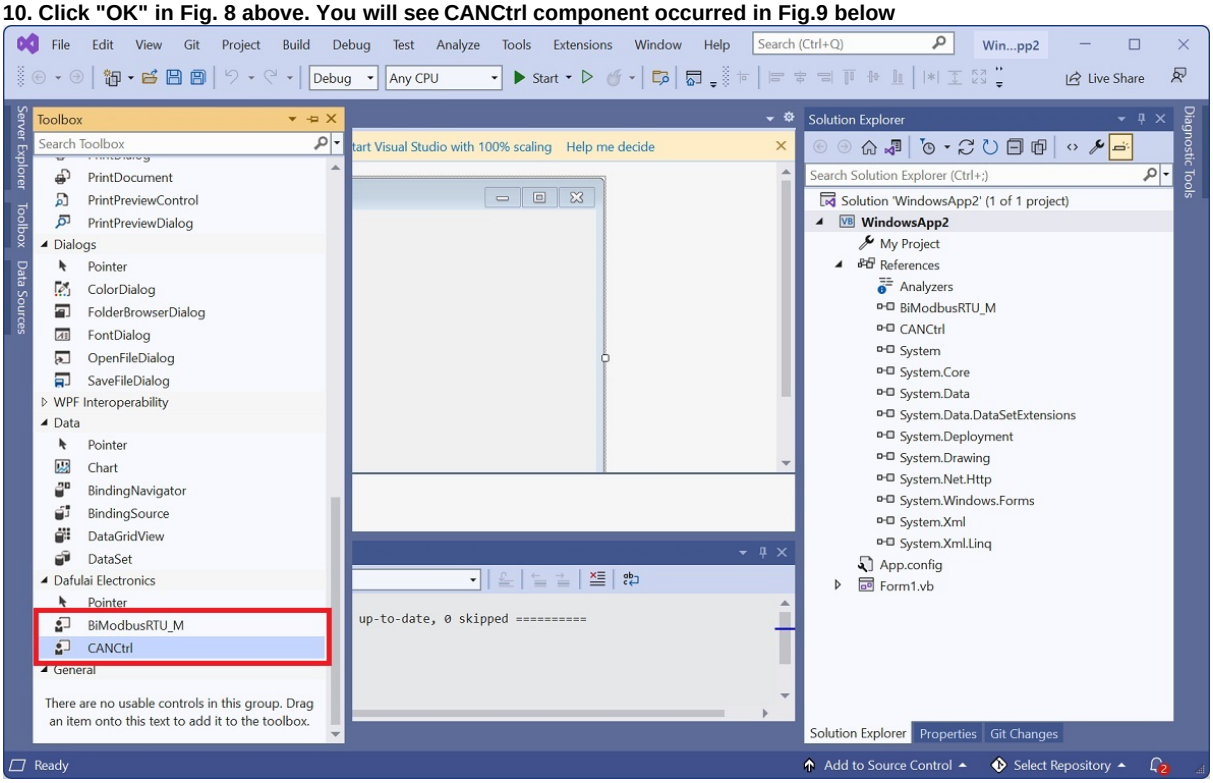

Fig.9 CANCtrl Component

11. Double click "CANCtrl" in Fig. 9 above. You will add component "CANCtrl" to your project in Fig.10 below.

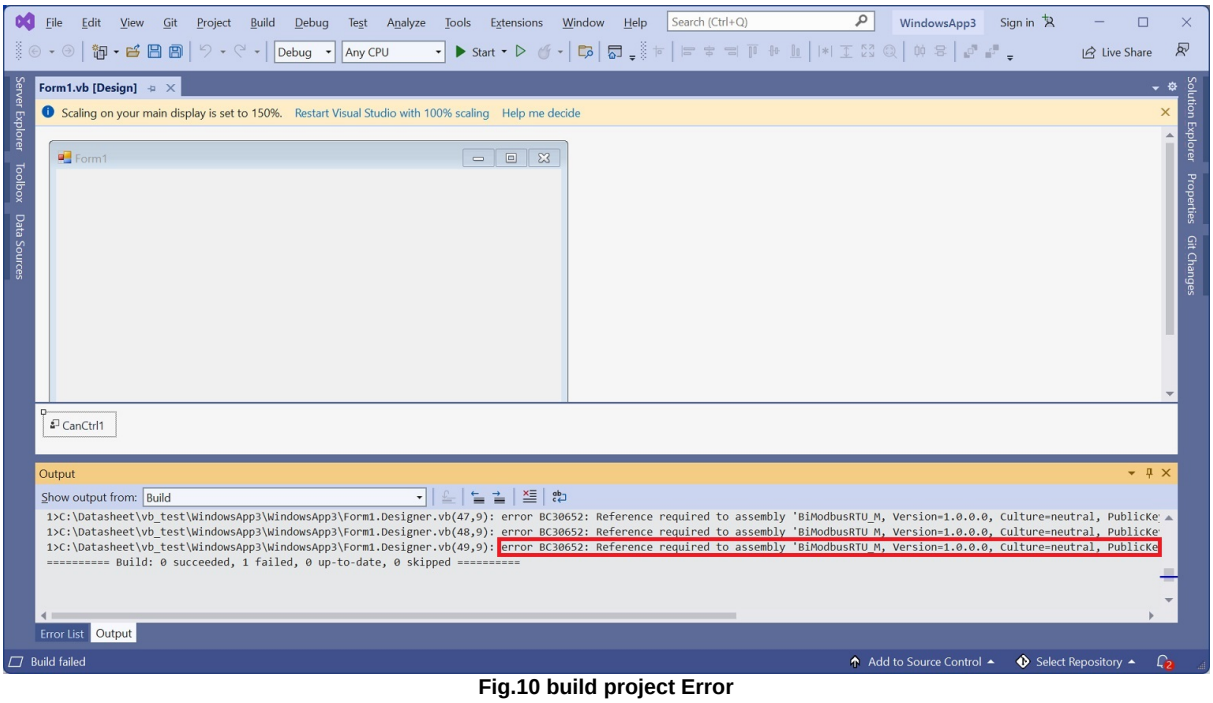

Fig.10 build project Error

12.Build your project in Fig.10 above, you will see error. Please add component by double clicking "BiModbusRTU_M" as Fig. 11. Remove component "BiModbusRTU_M" by "delete" key.

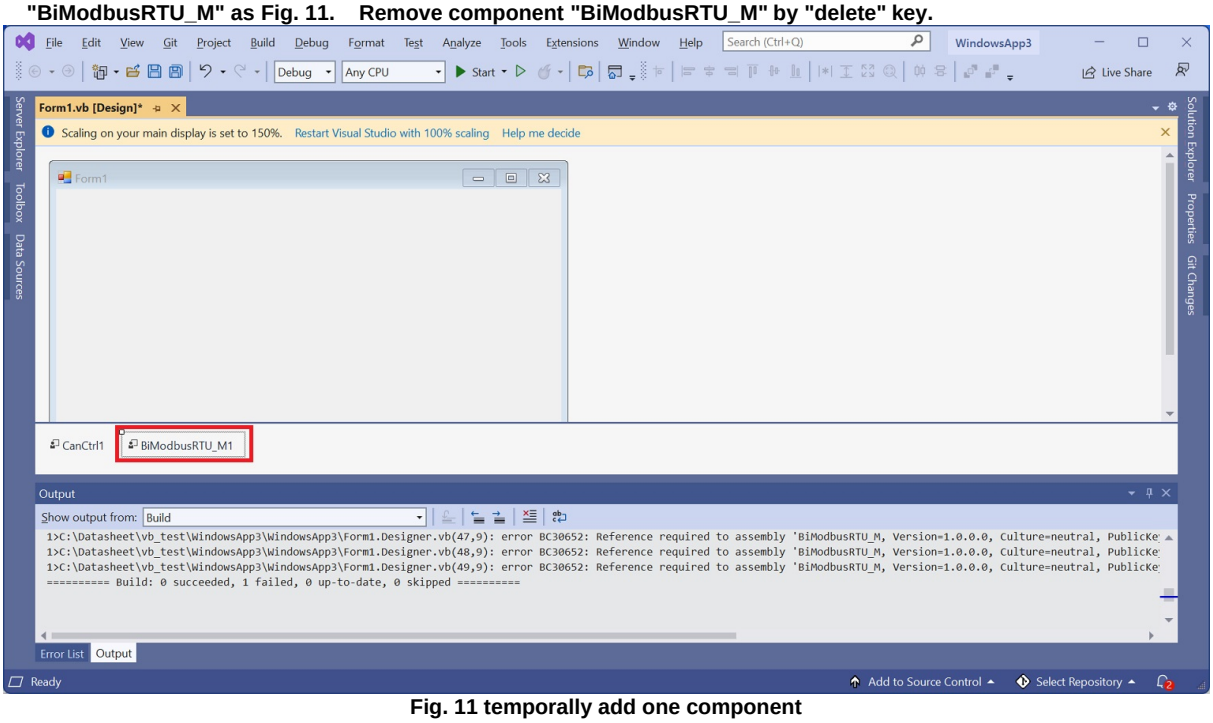

Fig. 11 temporally add one component

13. Build your project. it successes. Please see Fig.12 below

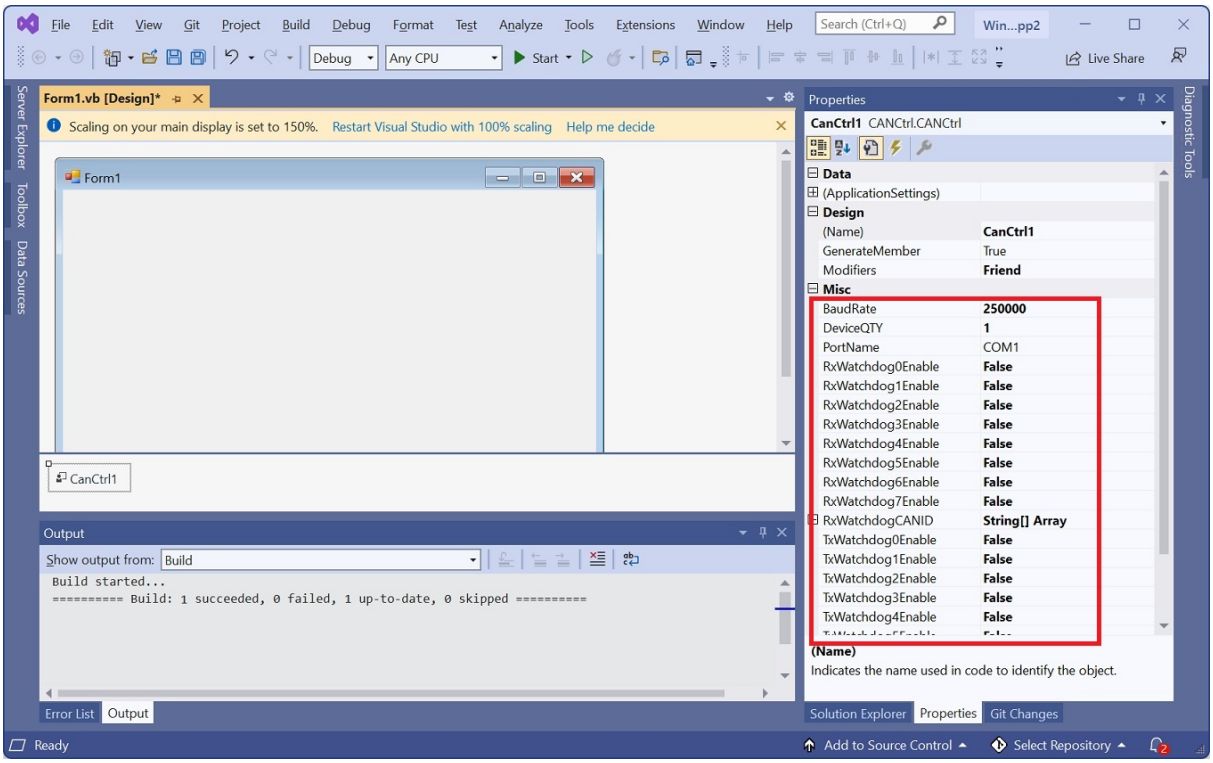

Fig.12 Properties

Now, we have installed CAN BUS component in Visual Studio IDE environmemt, You can see many properties for CANCtrl component in Fig. 12 inside red rectangle. In Fig.13 red rectangle, you will see 2 events for CANCtrl component.

You will use properties/methods/events from CANCtrl component to do everything you like.

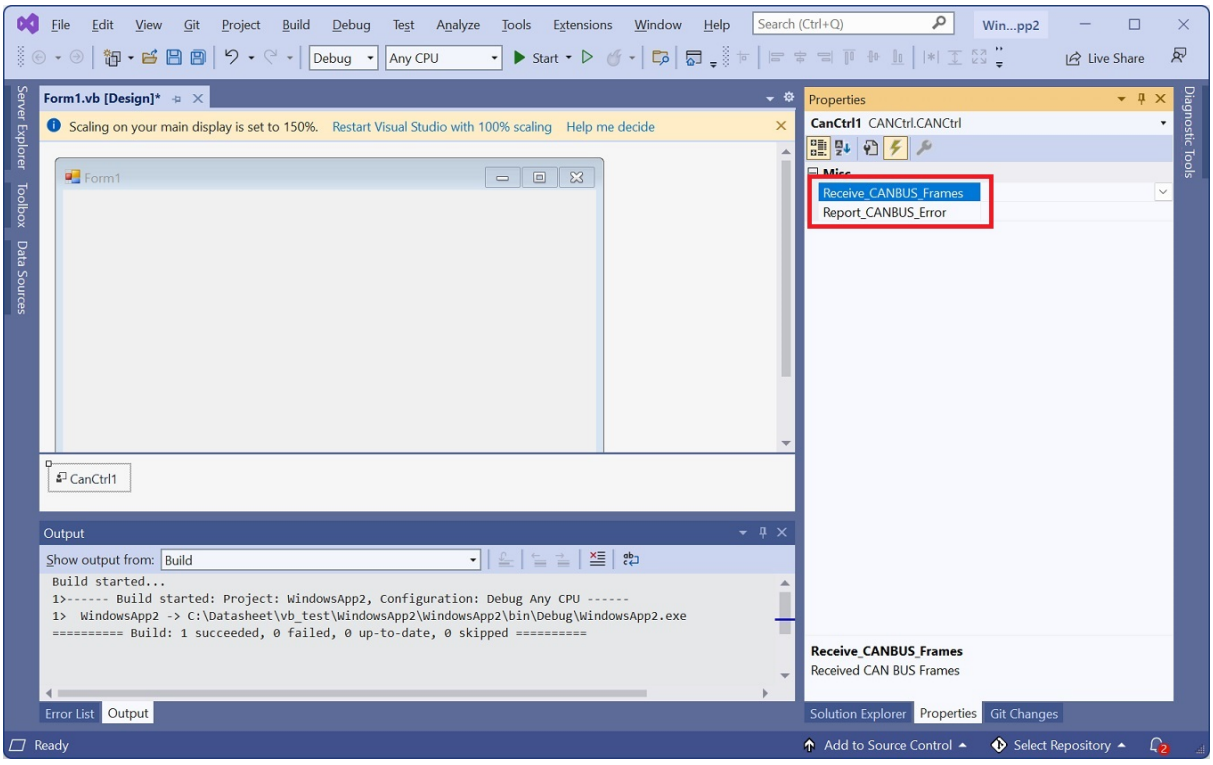

Fig.13 Events

3 Properties

All properities can be viewed in Fig.12 red rectangle.
We explain their details below:

- **PortName**

Daulai Electronics's CAN BUS analyzer/simulator will use USB to transmit/receive CAN bus data packets. Our Analyzer/Simulator uses one USB Serial Port.
This is Serial port name in String. Its value will be "COM1", "COM2", ...

- **BaudRate**

This is CAN BUS baud rate. Property value is Int32 data type, Its value is 25000 to 1000000 range. When BaudRate<50000, 1500000/BaudRate must be integer. If not, BaudRate will be rounded to keep this automatically. When BaudRate>=50000, 3000000/BaudRate must be integer. If not, BaudRate will be rounded to keep this automatically. This Property is readable and writable. So you can read back this property to verify.
If you set up BaudRate in run time, you must call Reset method before you set this property.

- **DeviceQTY**

Daulai Electronics's CAN BUS analyzer/simulator can simulate multiple CAN bus devices. This Property means the quantity of CAN bus devices you can simulate.
Property value is Int32 data type, Its value is 1,2,3,4,5,6,7 and 8. Each device you simulated has one transmit Watchdog and one receiver Watchdog. Watchdog is used in CAN BUS communication fault detection or synchronization. Entire CAN BUS only has one or zero watchdog as synchronization. Watchdog can be enabled/disabled. Please read TxWatcgdogCANID and RxWatcgdogCANID Property about watchdog principle. If you use CAN BUS analyzer, you can take transmit Watchdog as CAN bus transmitting data packet periodically by hardware timer. This Property is readable and writable.

- **TxWatchdogCANID**

This is CAN BUS CAN ID for transmitting watchdog, Property value is String array type. It has 8 elements. Each element will transmit periodically if enabled. We use hex ASCII string to denote CAN ID. If CAN ID is extended 29 bit ID, you must use "X" or "x" as suffix. For example, 29 bit CAN ID 0x34, you will put string value as 34x or 34X.
Prefix "0x" is option. This Property is readable and writable.

How does Watchdog work? We introduce RxWatchdog (Receiving Watchdog) firstly.
Each Device you simulate has one watchdog. Watchdog is ms counter, Counter length is called watchdog bytes quantity, And counter can be Up/Down counter and it is unsigned. Counter length can be configurable by methods. Counter direction (up/down) is decided by watchdog mode. Each watchdog has 3 different working mode.

Mode 0 : Watchdog initial value is 0. Watchdog is free running up-counter each ms Watchdog value will return 0 if RxWatchdog (receiver Watchdog) value is changed. When Watchdog value is over periods, it will keep the watch dog value and communication fault will occur for specified device.
Mode 1 :Watchdog initial value is 0. Watchdog is free running up-counter each ms. When it arrive at periods value, it will keep it, and one communication fault will occur for specified device. RxWatchdog Data packet (receiving watchdog data packet) can change counter value to avoid communication fault.
Mode 2 :Watchdog initial value is equal to periods. Watchdog is free running down-counter each ms. When it arrive at 0, it will keep 0 and communication fault will occur for specified device. RxWatchdog Data packet (receiving watchdog data packet) can change counter value to avoid communication fault.

You will setup RxWatchdog (Receiver watchdog) by properties and methods. RxWatchdog mode is Device watchdog mode. RxWatchdog period is device watchdog period which is used as communication Fault detection. When device received Rxwatchdog, it will use Rxwatchdog value to change Counter value.

For TxWatchdog (Transmitting Watchdog), it is used to avoid other CAN bus equipment communication fault. It has 4 different working mode.
Mode 0 : TxWatchdog value is decided by TxWatchdog-related method (Method: SetTxWatchdogV, or SetTxWatchdogV_Sync). Value 's position and Length in CAN BUS data packet is decided by TxWatchdog-related methods
Mode 1 : TxWatchdog value increases 1 every TxWatchdog's period automatically. You don't need to use SetTxWatchdogV or SetTxWatchdogV_Sync method to give TxWatchdog value. Value 's position and Length in CAN BUS data packet is decided by TxWatchdog-related methods. Value is unsigned. When value > maximum unsigned number, it will return 0.
Mode 2 : TxWatchdog value decreases 1 every TxWatchdog's period automatically. You don't need to use SetTxWatchdogV or SetTxWatchdogV_Sync method to give TxWatchdog value. Value 's position and Length in CAN BUS data packet is decided by TxWatchdog-related methods. Value is unsigned. When value = 0, it will return maximum unsigned number next time.
Mode 3 : TxWatchdog will have no any value, its data packet length is zero. It is used for CANOpen Sync frame.

In addition to avoid CAN BUS communication fault by using TxWatchdog, you can use TxWatchdog as CAN bus transmitting data packet periodically by hardware timer.

Notes: If you change only one TxWatchdogCANID on run time, you must assign entire 8 TxWatchdogCANID because this property is String array, not one String.

- **RxWatchdogCANID**

This is CAN BUS CAN ID for receiver watchdog, Property value is String array type. It has 8 elements. We use hex ASCII string to denote CAN ID. If CAN ID is extended 29 bit ID, you must use "X" or "x" as suffix. For example, 29 bit CAN ID 0x34, you will put string value as 34x or 34X.
Prefix "0x" is option. This Property is readable and writable. The watchdog principle is explained in "TxWatchdogCANID" property.

Notes: If you change only one RxWatchdogCANID on run time, you must assign entire 8 RxWatchdogCANID because this property is String array, not one String.

- **TxWatchdog0Enable**

This property is used for enable/disable TxWatchdog 0. Property value is Boolean type. True means enable, False means Disable. This Property is readable and writable.

- **TxWatchdog1Enable**

This property is used for enable/disable TxWatchdog 1. Property value is Boolean type. True means enable, False means Disable. This Property is readable and writable.

- **TxWatchdog2Enable**

This property is used for enable/disable TxWatchdog 2. Property value is Boolean type. True means enable, False means Disable. This Property is readable and writable.

- **TxWatchdog3Enable**

This property is used for enable/disable TxWatchdog 3. Property value is Boolean type. True means enable, False means Disable. This Property is readable and writable.

- **TxWatchdog4Enable**

This property is used for enable/disable TxWatchdog 4. Property value is Boolean type. True means enable, False means Disable. This Property is readable and writable.

- **TxWatchdog5Enable**

This property is used for enable/disable TxWatchdog 5. Property value is Boolean type. True means enable, False means Disable. This Property is readable and writable.

- **TxWatchdog6Enable**

This property is used for enable/disable TxWatchdog 6. Property value is Boolean type. True means enable, False means Disable. This Property is readable and writable.

- **TxWatchdog7Enable**

This property is used for enable/disable TxWatchdog 7. Property value is Boolean type. True means enable, False means Disable. This Property is readable and writable.

- **RxWatchdog0Enable**

This property is used for enable/disable RxWatchdog 0. Property value is Boolean type. True means enable, False means Disable. This Property is readable and writable.

- **RxWatchdog1Enable**

This property is used for enable/disable RxWatchdog 1. Property value is Boolean type. True means enable, False means Disable. This Property is readable and writable.

- **RxWatchdog2Enable**

This property is used for enable/disable RxWatchdog 2. Property value is Boolean type. True means enable, False means Disable. This Property is readable and writable.

- **RxWatchdog3Enable**

This property is used for enable/disable RxWatchdog 3. Property value is Boolean type. True means enable, False means Disable. This Property is readable and writable.

- **RxWatchdog4Enable**

This property is used for enable/disable RxWatchdog 4. Property value is Boolean type. True means enable, False means Disable. This Property is readable and writable.

- **RxWatchdog5Enable**

This property is used for enable/disable RxWatchdog 5. Property value is Boolean type. True means enable, False means Disable. This Property is readable and writable.

- **RxWatchdog6Enable**

This property is used for enable/disable RxWatchdog 6. Property value is Boolean type. True means enable, False means Disable. This Property is readable and writable.

- **RxWatchdog7Enable**

This property is used for enable/disable RxWatchdog 7. Property value is Boolean type. True means enable, False means Disable. This Property is readable and writable.

4 Methods

We describe all methods in VB.net syntax. You can easily use C# syntax to call it.

- **Function Open() As Boolean**

It will Enable CAN BUS. Its actions are below:

1. Open serial port if it is not opened.
2. Put all Settings in PC by methods or properties into our CAN BUS board. It means that all parameters values are in PC file before you call Open method.
3. Enable CAN bus function. It means that CAN Bus board can send acknowledge bit to any CAN bus frame it received

In above 3 actions, any failure will return false. Otherwise return true. In general, when we want to use CAN Bus , we should call open() method firstly after finish all parameters settings.

- **Function ResetMCU() As Boolean**

It will make CAN BUS board reset, just like CAN BUS board re-power on again. So CAN BUS board initial state will be guaranteed. It means that CAN BUS is disabled, and CAN Bus data packet's timestamp zero point is here.

This method will return true if success, otherwise return false. In general, when we run your application software, you use this method to make sure CAN BUS is disabled and timestamp=0 at beginning.

- **Sub Close()**

It will Disable CAN BUS. It means that CAN Bus board can not send acknowledge bit to any CAN bus frame it received.

- **Sub CloseCOM()**

It will close serial port. Usually you call it 50 ms later after you call Close method. You don't need call it if your purpose for calling Close method is for disable CAN BUS. You use Close method as possible as you can except you want to change PortName.

- **Function IsOpen() As Boolean**

It will return true if CAN Bus is enabled, otherwise return false

- **Function ReadHardwareConfig() As Boolean**

It will read CAN BUS board all settings and give them to PC. It will return true if readings success,

otherwise return false.

Notes:1 If you use this method, it will change properties settings in the VS design window.

2. All methods's reading back value except this method, and property's reading value are from PC instead of Hardware. So you want these values from hardware, you must call this method firstly.

- **Function CANtransmit(CANID As UInteger) As Result**

It will transmit remote CAN bus frame with CAN ID= CANID. This is Non-Block function.

CANID is input parameter , type is unsigned 32bits integer. If it is extended CAN ID, you must use bit OR operation with CAN_ID_Type.Extended constant.

For example, 29bit ID= 0x1867ffff , you must use &H1867ffff Or CAN_ID_Type.Extended as input parameter in VB.net

Return value is enum Result. When return value is constant: Result.CONTINUE_EXE, you must call it again at proper time later.

If return value is Result.SUCCESS, it means this remote frame transmitting request successes.

Notes: Result.SUCCESS does not mean CAN BUS transmitting successfully, it just mean CAN Bus board received PC request successfully. How to know transmitting successes ?

You must use Analyzer mode, and from CAN BUS receive event to know this received frame is transmitted frame.

- **Function CANtransmit_Sync(CANID As UInteger) As Boolean**

It is the same as above method except this is block function.

It will return true if transmitting request successes, otherwise return false.

Notes: true does not mean CAN BUS transmitting successfully, it just mean CAN Bus board received PC request successfully. *How to know transmitting successes ?*

You must use Analyzer mode, and from CAN BUS receive event to know this received frame is transmitted frame.

- **Function CANtransmit(CAN_FrameQty As SByte , CANID() As UInteger) As Result**

It will transmit multiple remote CAN bus frames with CAN ID= CANID array. This is Non-Block function.

The 1st Input parameter: CAN_FrameQty, type is signed byte, it denotes how many remote frames will be sent.

The 2nd Input parameter: CANID, type is unsigned 32bits integer array, it denotes remote frames' CAN ID. Similarly, If it is extended CAN ID, you must use bit OR operation with CAN_ID_Type.

Extended constant.
Return value is enum Result. When return value is constant: Result.CONTINUE_EXE, you must call it again at proper time later.
If return value is Result.SUCCESS, it means this remote frame transmitting request successes.

- **Function CANtransmit_Sync(CAN_FrameQty As SByte , CANID() As UInteger) As Boolean**

It is the same as above method except this is block function.
It will return true if transmitting request successes, otherwise return false.

- **Function CANtransmit(CANID As UInteger , DLC As Byte , Data() As Byte) As Result**

It will transmit CAN bus Data frame with CAN ID= CANID and Data length=DLC and Data byte = Data Array. This is Non-Block function.
The 1st Input parameter: CANID , type is unsigned 32bits integer. it denotes data frames' CAN ID. If it is extended CAN ID, you must use bit OR operation with CAN_ID_Type.Extended constant.
The 2nd Input parameter: DLC, type is byte. it denotes length of data.
The 3rd Input parameter: Data(), type is byte array. it denotes specific data value.
Return value is enum Result. When return value is constant: Result.CONTINUE_EXE, you must call it again at proper time later.
If return value is Result.SUCCESS, it means this data frame transmitting request successes.

- **Function CANtransmit_Sync(CANID As UInteger , DLC As Byte , Data() As Byte) As Boolean**

It is the same as above method except this is block function.
It will return true if transmitting request successes, otherwise return false.

- **Function CANtransmit(CAN_FrameQty As SByte, CANID() As UInteger, DLC () as Byte, Data()() as byte) As Result**

It will transmit a group of CAN bus data frames with CAN ID= CANID array and Data length=DLC array and Data byte = Data()() Array. This is Non-Block function.
The 1st Input parameter: CAN_FrameQty, type is signed byte, it denotes how many data frames will be sent.
The 2nd Input parameter: CANID() , type is unsigned 32bits integer array. it denotes data frames' CAN ID. If it is extended CAN ID, you must use bit OR operation with CAN_ID_Type.Extended constant.

The 3rd Input parameter: DLC(), type is byte array. it denotes length of data.
The 4th Input parameter: Data()(), type is 2- dimension byte array. it denotes specific data value. The first index of array denotes frame number.
Return value is enum Result. When return value is constant: Result.CONTINUE_EXE, you must call it again at proper time later.
If return value is Result.SUCCESS, it means this data frame transmitting request successes

- **Function CANtransmit_Sync(CAN_FrameQty As SByte, CANID() As UInteger, DLC() As Byte, Data()() As Byte) as Boolean**

It is the same as above method except this is block function.
It will return true if transmitting request successes, otherwise return false.

- **Sub SetFilter(FilterNo As Byte, FilterValue As UInteger)**

Before we describe filter-related / Mask-related method, we explain CAN bus receive principle.
What CAN bus ID we can receive depends on what filters/masks we set.
Our CAN bus hardware has 16 Filters (0 to 15), 3 Masks (0 to 2). Every Filter can choose one of 3 Masks.
Our CAN bus hardware can receive CAN frame with VB.net statement below
(CAN ID) And Mask = Filter And Mask
What does it mean for above statement? CAN BUS only receive CAN ID = Filter, and if related Mask bit is zero, CAN Bus receiver does not care related bit of CAN ID whether it is equal to related bit of filter.
Each Filter can be enabled or disabled. If you want to receive extended frame, you must make filter bit or operation with constant CAN_ID_Type.Extended.

Now, we describe method: SetFilter(FilterNo as Byte, FilterValue As UInteger)
It will set filter's value.
The 1st Input parameter: FilterNo, type is Byte, it denotes filter number (0 to 15).
The 2nd Input parameter: FilterValue, type is UInteger, it denotes filter value.

- **Sub EnableFilter(FilterNo As Byte)**

It will enable filter.
The 1st Input parameter: FilterNo, type is Byte, it denotes filter number (0 to 15).

- **Sub DisableFilter(FilterNo As Byte)**

It will disable filter.
The 1st Input parameter: FilterNo, type is Byte, it denotes filter number (0 to 15).

- **Function GetFilter(FilterNo As Byte) As UInteger**

It will get filter value.

The 1st Input parameter: FilterNo, type is Byte, it denotes filter number (0 to 15).

Return value is filter value, type is UInteger.

You will use the following VB.Net to get whether it is Extended 29 bit ID

```
if (GetFilter(FilterNo) And CAN_ID_Feature.Mask4IDType)=CAN_ID_Type.Extended Then
    ' Extended CAN ID
Else
    'Standard CAN ID
End If
```

The truly CAN ID value is statement in VB.net below:

```
GetFilter(FilterNo) And CAN_ID_Feature.Mask4Got29BitID
```

Or

```
GetFilter(FilterNo) And CAN_ID_Feature.Mask4Got11BitID
```

- **Sub SetMask(MaskNo As Byte, MaskValue As UInteger)**

It will set Mask's value.

The 1st Input parameter: MaskNo, type is Byte, it denotes Mask number (0 to 2).

The 2nd Input parameter: MaskValue, type is UInteger, it denotes MaskValue value.

- **Function GetMask(MaskNo As Byte) As UInteger**

It will get Mask value.

The 1st Input parameter: MaskNo, type is Byte, it denotes Mask number (0 to 2).

Return value is Mask value, type is UInteger.

- **Sub SetMaskSelect4Filter(Mask4Filter0 As Byte, Mask4Filter1 As Byte, ..., Mask4Filter15 As Byte)**

It will set Filter's Mask number to use, it can be only one input parameter for setting up filter0's Mask number (0 to 2), or it can be as many as 16 input parameters to set up all filters' mask number (0 to 2)

The 1st Input parameter: Mask4Filter0, type is Byte, it denotes Mask number (0 to 2) Filter0 used.

The 2nd Input parameter: Mask4Filter1, type is Byte, it denotes Mask number (0 to 2) Filter1 used.

The 3rd Input parameter: Mask4Filter2, type is Byte, it denotes Mask number (0 to 2) Filter2 used.
The 4th Input parameter: Mask4Filter3, type is Byte, it denotes Mask number (0 to 2) Filter3 used.
The 5th Input parameter: Mask4Filter4, type is Byte, it denotes Mask number (0 to 2) Filter4 used.
The 6th Input parameter: Mask4Filter5, type is Byte, it denotes Mask number (0 to 2) Filter5 used.
The 7th Input parameter: Mask4Filter6, type is Byte, it denotes Mask number (0 to 2) Filter6 used.
The 8th Input parameter: Mask4Filter7, type is Byte, it denotes Mask number (0 to 2) Filter7 used.
The 9th Input parameter: Mask4Filter8, type is Byte, it denotes Mask number (0 to 2) Filter8 used.
The 10th Input parameter: Mask4Filter9, type is Byte, it denotes Mask number (0 to 2) Filter9 used.
The 11th Input parameter: Mask4Filter10, type is Byte, it denotes Mask number (0 to 2) Filter10 used.
The 12th Input parameter: Mask4Filter11, type is Byte, it denotes Mask number (0 to 2) Filter11 used.
The 13th Input parameter: Mask4Filter12, type is Byte, it denotes Mask number (0 to 2) Filter12 used.
The 14th Input parameter: Mask4Filter13, type is Byte, it denotes Mask number (0 to 2) Filter13 used.
The 15th Input parameter: Mask4Filter14, type is Byte, it denotes Mask number (0 to 2) Filter14 used.
The 16th Input parameter: Mask4Filter15, type is Byte, it denotes Mask number (0 to 2) Filter15 used.

- **Function GetMaskSelect4Filter(FilterNo As Byte) As Byte**

It will get Mask number Filter used.
The 1st Input parameter: FilterNo, type is Byte, it denotes filter number (0 to 15).
Return value is Mask Number, type is Byte (0 to 2).

- **Sub SetTerminator(Enable As Boolean)**

It will enable/disable CAN BUS 120 Ohms terminator.
The 1st Input parameter: Enable, type is Boolean. If True, we will enable 120 Ohms terminator, otherwise, Disable 120 Ohms terminator

- **Function GetTerminator() As Boolean**

It will get whether CAN BUS 120 Ohms terminator is used.
Return value: True if 120 Ohms terminator used, False if 120 Ohms terminator did not use.

- **Sub SetRxWatchdogDLC(DeviceNo As Byte, Len As Byte)**

It will set up Receiver Watchdog CAN Bus Data Frame Length.

The 1st Input parameter: DeviceNo, type is Byte, it denotes Rx Watchdog number (0 to 7). It only has meaning for 0 to Property: DeviceQTY-1

The 2nd Input parameter: Len, type is Byte, it denotes Rx Watchdog frame's data length (1 to 8). It is not watchdog value's length.

- **Function GetRxWatchdogDLC(DeviceNo As Byte) As Byte**

It will set up Receiver Watchdog CAN Bus Data Frame Length.

The 1st Input parameter: DeviceNo, type is Byte, it denotes Rx Watchdog number (0 to 7). It only has meaning for 0 to Property: DeviceQTY-1

Return value is Rx Watchdog frame's data length, type is Byte (1 to 8). It is not watchdog value's length.

- **Sub SetRxWatchdogValuePosition(DeviceNo As Byte, Position As Byte)**

It will set up Receiver Watchdog value's start position (1-based) in the receiver Watchdog data frame. (Because not all data in the frame are the watchdog value).

The 1st Input parameter: DeviceNo, type is Byte, it denotes Rx Watchdog number (0 to 7). It only has meaning for 0 to Property: DeviceQTY-1

The 2nd Input parameter: Position, type is Byte, it denotes Rx Watchdog value's start position (1-based) in the receiver Watchdog data frame. its value is 1 to 8

- **Function GetRxWatchdogValuePosition(DeviceNo As Byte) As Byte**

It will get Receiver Watchdog value's start position (1-based) in the receiver Watchdog data frame. (Because not all data in the frame are the watchdog value).

The 1st Input parameter: DeviceNo, type is Byte, it denotes Rx Watchdog number (0 to 7). It only has meaning for 0 to Property: DeviceQTY-1

Return value is Rx Watchdog value's start position in frame's data field, type is Byte (1 to 8). .

- **Sub SetRxWatchdogValueLength(DeviceNo As Byte, Length As Byte)**

It will set up Receiver Watchdog value's byte quantities in the receiver Watchdog data frame. (Because not all data in the frame are the watchdog value).

The 1st Input parameter: DeviceNo, type is Byte, it denotes Rx Watchdog number (0 to 7). It only has meaning for 0 to Property: DeviceQTY-1

The 2nd Input parameter: Length, type is Byte, it denotes Rx Watchdog value's byte quantites in the receiver Watchdog data frame. its value is 1 or 2 or 4 or 8

- **Function GetRxWatchdogValueLength(DeviceNo As Byte) As Byte**

It will get Receiver Watchdog value's byte quantities in the receiver Watchdog data frame. (Because

not all data in the frame are the watchdog value).

The 1st Input parameter: DeviceNo, type is Byte, it denotes Rx Watchdog number (0 to 7). It only has meaning for 0 to Property: DeviceQTY-1

Return value is Rx Watchdog value's byte quantities in the receiver Watchdog data frame. Type is Byte. Its value is 1 or 2 or 4 or 8

- **Sub SetRxWatchdogMode(DeviceNo As Byte, Mode As Byte)**

It will set up Receiver Watchdog mode.

The 1st Input parameter: DeviceNo, type is Byte, it denotes Rx Watchdog number (0 to 7). It only has meaning for 0 to Property: DeviceQTY-1

The 2nd Input parameter: Mode, type is Byte, it denotes Rx Watchdog mode. its value is 0, 1, and 2.

- **Function GetRxWatchdogMode(DeviceNo As Byte) As Byte**

It will get Receiver Watchdog mode.

The 1st Input parameter: DeviceNo, type is Byte, it denotes Rx Watchdog number (0 to 7). It only has meaning for 0 to Property: DeviceQTY-1

Return value is Rx Watchdog modee. Type is Byte. its value is 0, 1, and 2.

- **Sub SetRxWatchdogPeriod(DeviceNo As Byte, Period As UShort)**

It will set up Receiver Watchdog Period in ms.

The 1st Input parameter: DeviceNo, type is Byte, it denotes Rx Watchdog number (0 to 7). It only has meaning for 0 to Property: DeviceQTY-1

The 2nd Input parameter: Periods in ms, type is unsigned 16 bit integer.

- **Sub SetTxWatchdogDLC(DeviceNo As Byte, Len As Byte)**

It will set up Transmitter Watchdog CAN Bus Data Frame Length.

The 1st Input parameter: DeviceNo, type is Byte, it denotes Tx Watchdog number (0 to 7). It only has meaning for 0 to Property: DeviceQTY-1

The 2nd Input parameter: Len, type is Byte, it denotes Tx Watchdog frame's data length (1 to 8). It is not watchdog value's length.

- **Function GetTxWatchdogDLC(DeviceNo As Byte) As Byte**

It will set up Transmitter Watchdog CAN Bus Data Frame Length.

The 1st Input parameter: DeviceNo, type is Byte, it denotes Tx Watchdog number (0 to 7). It only has meaning for 0 to Property: DeviceQTY-1

Return value is Tx Watchdog frame's data length, type is Byte (1 to 8). It is not watchdog value's length.

- **Sub SetTxWatchdogValuePosition(DeviceNo As Byte, Position As Byte)**

It will set up Transmitter Watchdog value's start position (1-based) in the Transmitter Watchdog data frame. (Because not all data in the frame are the watchdog value).

The 1st Input parameter: DeviceNo, type is Byte, it denotes Tx Watchdog number (0 to 7). It only has meaning for 0 to Property: DeviceQTY-1

The 2nd Input parameter: Position, type is Byte, it denotes Tx Watchdog value's start position (1-based) in the Transmitter Watchdog data frame. its value is 1 to 8

- **Function GetTxWatchdogValuePosition(DeviceNo As Byte) As Byte**

It will get Transmitter Watchdog value's start position (1-based) in the Transmitter Watchdog data frame. (Because not all data in the frame are the watchdog value).

The 1st Input parameter: DeviceNo, type is Byte, it denotes Tx Watchdog number (0 to 7). It only has meaning for 0 to Property: DeviceQTY-1

Return value is Tx Watchdog value's start position in frame's data field, type is Byte (1 to 8). .

- **Sub SetTxWatchdogValueLength(DeviceNo As Byte, Length As Byte)**

It will set up Transmitter Watchdog value's byte quantities in the Transmitter Watchdog data frame. (Because not all data in the frame are the watchdog value).

The 1st Input parameter: DeviceNo, type is Byte, it denotes Tx Watchdog number (0 to 7). It only has meaning for 0 to Property: DeviceQTY-1

The 2nd Input parameter: Length, type is Byte, it denotes Tx Watchdog value's byte quantities in the Transmitter Watchdog data frame. its value is 1 or 2 or 4 or 8

Notes: If txWatchdog Mode=3, transmitter Watchdog value's byte quantities will be zero, and ignore this method's setting value

- **Function GetTxWatchdogValueLength(DeviceNo As Byte) As Byte**

It will get Transmitter Watchdog value's byte quantities in the Transmitter Watchdog data frame. (Because not all data in the frame are the watchdog value).

The 1st Input parameter: DeviceNo, type is Byte, it denotes Tx Watchdog number (0 to 7). It only has meaning for 0 to Property: DeviceQTY-1

Return value is Tx Watchdog value's byte quantities in the Transmitter Watchdog data frame. Type is Byte. Its value is 1 or 2 or 4 or 8

- **Sub SetTxWatchdogMode(DeviceNo As Byte, Mode As Byte)**

It will set up Transmitter Watchdog mode.

The 1st Input parameter: DeviceNo, type is Byte, it denotes Tx Watchdog number (0 to 7). It only has meaning for 0 to Property: DeviceQTY-1
The 2nd Input parameter: Mode, type is Byte, it denotes Tx Watchdog mode. its value is 0, 1, 2, and 3.

- **Function GetTxWatchdogMode(DeviceNo As Byte) As Byte**

It will get Transmitter Watchdog mode.
The 1st Input parameter: DeviceNo, type is Byte, it denotes Tx Watchdog number (0 to 7). It only has meaning for 0 to Property: DeviceQTY-1
Return value is Tx Watchdog modee. Type is Byte. its value is 0, 1, 2, and 3.

- **Sub SetTxWatchdogPeriod(DeviceNo As Byte, Period As UShort)**

It will set up Transmitter Watchdog Period in ms.
The 1st Input parameter: DeviceNo, type is Byte, it denotes Rx Watchdog number (0 to 7). It only has meaning for 0 to Property: DeviceQTY-1
The 2nd Input parameter: Periods in ms, type is unsigned 16 bit integer.

- **Function SetTxWatchdogV(DeviceNo As Byte, Value As Byte()) As Result**

It will set up Transmitter Watchdog Value. It will send to Hardware directly. This is Non-block function. It is only useful for mode 0.
The 1st Input parameter: DeviceNo, type is Byte, it denotes Tx Watchdog number (0 to 7). It only has meaning for 0 to Property: DeviceQTY-1
The 2nd Input parameter: Value, type is Byte array, it denotes Tx Watchdog value which is Little-endian.
Return value is enum Result. When return value is constant: Result.CONTINUE_EXE, you must call it again at proper time later.
If return value is Result.SUCCESS, it means this data frame transmitting request successes

Notes: If you want to call this method to set TxWatchdog Value, you must be after all TxWachdog Value's length been set. And you must be after open() because length setting is only set to hardware after open. Of cause if you didn't change TxWachdog Value's length, you can call this method at any time. Otherwise, unexpected result will occur.

- **Function SetTxWatchdogV_Sync(DeviceNo As Byte, Value As Byte()) As Boolean**

It will set up Transmitter Watchdog Value. It will send to Hardware directly. This is block function.It is

only useful for mode 0.

The 1st Input parameter: DeviceNo, type is Byte, it denotes Tx Watchdog number (0 to 7). It only has meaning for 0 to Property: DeviceQTY-1

The 2nd Input parameter: Value, type is Byte array, it denotes Tx Watchdog value which is Little-endian.

Return value True means success , False means failure.

- **Sub SetTxSync(DeviceNo As Byte, EnableSync As Boolean)**

It will Enable/Disable Tx Wachdog synchronization. Synchronization means all transmitting of CAN Bus frames must happen after transmitting this Tx Watchdog.

We only allow one TxWachdog as synchronous frame. When enable new Tx Watchdog as synchronous frame, the old synchronous Tx Watchdog frame will become disable synchronization automatically.

The 1st Input parameter: DeviceNo, type is Byte, it denotes Tx Watchdog number (0 to 7). It only has meaning for 0 to Property: DeviceQTY-1

The 2nd Input parameter: EnableSync, type is Byte Boolean, True denotes this Tx Watchdog synchronization is enabled.

- **Sub SetToAnalyzer( )**

It will set this CAN BUS tool as Analyzer which means you can receive what you transmit. (Echo On)

If you purchase "Simulator only", you cannot set to Analyzer.

Notes: These setting is only to put into PC file, didn't put into Hardware. You must call Open method to enter hardware at last

- **Sub SetToSimulator( )**

It will set this CAN BUS tool as simulator which means you cannot receive what you transmit.(Echo Off)

If you purchase "Analyzer only", you cannot set to Simulator.

Notes: These setting is only to put into PC file, didn't put into Hardware. You must call Open method to enter hardware at last

- **Function IsAnalyzer( ) As Boolean**

If CAN BUS tool is Analyzer (Echo On), it will return True , otherwise return False.

Notes: This judgment is only according to PC file, not according to actual hardware. You

must call ReadHardwareConfig method to put Hardware configuration into PC file before you call IsAnalyzer( ) method if you want to know hardware actual situation.

- **Function IsSimulator( ) As Boolean**

If CAN BUS tool is Simulator (Echo Off), it will return True , otherwise return False.

Notes: This judgment is only according to PC file, not according to actual hardware. You must call ReadHardwareConfig method to put Hardware configuration into PC file before you call IsSimulator( ) method if you want to know hardware actual situation.

- **Sub Load(FileName as String)**

It will load all properties and methods setting values to PC from file. Note: these values are to PC not to hardware.

The 1st Input parameter: FileName, type is String, it denotes directory and file name which contains hardware settings.

- **Sub Save(FileName as String)**

It will save all properties and methods setting values from PC to file. Note: these values are from PC not from hardware.

The 1st Input parameter: FileName, type is String, it denotes directory and file name which will save to.

- **Function WriteHardwareConfig() As Boolean**

It will put all properties and methods setting values from PC to hardware. Note: these values are only into hardware volatile memory, not flash.

Return true if success

- **Function StoreICConfigToFlash() As Boolean**

It will put all properties and methods setting values from hardware Volatile RAM to flash .

Return true if success

5 Events

- **Receive_CANBUS_Frames(FrameQty As UShort, CANID() As UInteger, Remote() As Boolean, DLC() as Byte, Data()() As Byte, timestamp() as UInteger)**

This event occurs when CAN BUS gets data frame or remote frame.
The 1st parameter: FrameQty is received CAN bus frame quantities, type is UShort.
The 2nd parameter: CANID() is received CAN ID , type is UInteger Array. Index is frame number which is from 0 to FrameQty-1. If you want to know whether it is extended frame, you must use the following VB.Net to get whether it is Extended 29 bit ID

```
If  (CANID(i) And CAN_ID_Feature.Mask4IDType)=CAN_ID_Type.Extended  Then
    ' Extended CAN ID

Else
    ' Standard CAN ID

End If
```

If you want to know whether it is my transmitting frame (self transmit, self receive), you must use the following VB.Net

```
If (CANID(i) And CAN_ID_Feature.Transmitted)=CAN_ID_Feature.Transmitted Then
    ' Echo Back frame (Self transmit, self receive)

Else
    ' outside frame receives

End If
```

The truly CAN ID value is statement in VB.net below:

```
CANID(i) And CAN_ID_Feature.Mask4Got29BitID
```

Or

```
CANID(i) And CAN_ID_Feature.Mask4Got11BitID
```

The 3rd parameter: Remote() is RTR flag , type is Boolean Array. Index is frame number which is from 0 to FrameQty-1. True means Remote Frame. False means Data Frame.
The 4th parameter: DLC() is Data Length , type is Byte Array. Index is frame number which is from 0 to FrameQty-1. Its value is 0 to 8
The 5th parameter: Data()(), type is 2- dimension byte array. it denotes specific data value. The first index of array denotes frame number which is from 0 to FrameQty-1.
The 6th parameter: timestamp() is time stamp of receiver, type is UInteger Array. Index is frame number which is from 0 to FrameQty-1. Scale is 0.125us /bit. Total 27 bits.
Time Stamp is zero when executing Method ResetMCU() or Power on Hardware.

This event will be running in different thread. So if you want to change control of Form, you must use delegate.
For example, there is Label control In your Form, you want to change Label.text = Data(0)(0) inside this event handle.
You must define a delegate by statement below:

```
Delegate Sub LabelTextSetting_Delegate(Byval [Label] As Label, Byval [Text] As String)
```

And you define a subroutine for setting Label Text below:

```
Private Sub setLabelText_ThreadSafe(Byval [Label] as Label, Byval [Text] as String)
    If [Label].InvokeRequired Then
Dim myDelegate As New LabelTextSetting_Delegate(AddressOf setLabelText_ThreadSafe)
        Me.Invoke(myDelegate, New Object(){[Label],[Text]})
    Else
        [Label].Text=[Text]
    End If
End Sub
```

If your Label name is "Label1", You just use this statement below to set Label1's text to Data(0)(0) below in the event handle:

```
setLabelText_ThreadSafe(Label1, CStr(Data(0)(0)) )
```

***Notes**: In some VS studio version, Receive_CANBUS_Frames event handle subroutine created automatically by IDE will cause "do not have a compatible signature" error when compiling. This is caused by argument Data type. Data*

wrongly used " Data As Byte()" by IDE, Please manually change to "Data As Byte()()" or "Data()() As Byte"

- **Report_CANBUS_Error(RxErrCount As Byte, TxErrCount As Byte, BusOFF As Boolean, CANBUS_Rxd_Overflow As Boolean, PC_Txd_Overflow As Boolean,HW_Memory4CANRxd_Overflow As Boolean, DeviceCOMFaultFlag As Byte)**

This is Error report event.
The 1st parameter: RxErrCount is CAN Bus receiver Error Counter, type is Byte.
The 2nd parameter: TxErrCount is CAN Bus Transmitter Error Counter, type is Byte. .
The 3rd parameter: BusOFF is CAB Bus OFF fault, type is Boolean. True means CAN Bus OFF.
The 4th parameter: CANBUS_Rxd_Overflow is that Hardware receives too many frames, but PC cannot receive them due to PC USB UART overflow. Type is Boolean. True means overflow occurs. It is not easy to occur because UART baud rate= 3M bit/sec, but maximum CAN BUS baud rate is 1M bit/sec
The 5th parameter: PC_Txd_Overflow is that PC transmits too many CAN BUS transmitting packet data to Hardware, but Hardware cannot hold so many data. True means overflow occurs. You can avoid this situation by slowly telling hardware data packets.
The 6th parameter: DeviceCOMFaultFlag is that CAN Bus communication fault flag according to RxWatchdog. Bit y is CAN BUS communication fault which is caused by the RxWatchdog number y (y=0,1,2,3,4,5,6,7). Value 1 means fault, 0 means normal.

This event is the same as Can bus receiver event, it will be running in different thread. So if you want to change control of Form, you must use delegate.

IMPORTANT NOTICE

The information in this manual is subject to change without notice.

Dafulai's products are not authorized for use as critical components in life support devices or systems. Life support devices or systems are those which are intended to support or sustain life and whose failure to perform can be reasonably expected to result in a significant injury or death to the user. Critical components are those whose failure to perform can be reasonably expected to cause failure of a life support device or system or affect its safety or effectiveness.

COPYRIGHT