CAN Bus Analyzer/Simulator for Matlab/Simulink

User Manual

Table of Contents

1 Introduce

Our CAN Bus Analyzer/Simulator Hardware can be used under Matlab/Simulink platform (Matlab 2021 or higher) for windows/Linux/mac OS.
We call Analyzer/Simulator CAN BUS Controller in Matlab/Simulink.
Let us introduce CAN Bus Controller function.
Our CAN Bus controller can transmit/Receive standard or/and extended CAN Bus data and RTR frames as general CAN bus node.
And furthermore, transmitting CAN Bus Frame can be synchronized by special CAN BUS transmitting frame or receiving frame. If Sync is enabled, CAN Bus controller only can transmit CAN Bus frame when it receives or transmits this special CAN BUS frame. This special CAN BUS frame is called "Sync frame"
Another feature is that our CAN Bus Controller has Watchdog function.
Watchdog purpose is for getting communication fault information.
Received Watchdog (Call it RxWatchdog) is for itself to know whether CAN BUS communication OK or not.
Transmitted Watchdog (Call it TxWatchdog) is for other CAN bus nodes to know whether CAN BUS communication OK or not.
TxWatchdog CAN Bus frame send out periodically automatically by hardware, You don't need to send by calling transmit method or block.
Similarly, if "Sync frame" is "Transmit frame", "Sync frame" send out periodically automatically by hardware, You don't need to send by calling transmit method or block.

RxWatchdog has 3 kinds of work modes:

1. Work mode 0: RxWatchdog initial value is 0. RxWatchdog is free running up-counter each ms. RxWatchdog value will return 0 if we receive Watchdog CAN Bus frame, and value is different from previous received value. When RxWatchdog value is over RxWatchdog Period, it will keep the Rxwatchdog value and communication fault will occur.
2. Work mode 1: RxWatchdog initial value is 0. RxWatchdog is free running up-counter each ms. When it arrive at period value, it will keep it, and one communication fault will occur. RxWatchdog Data packet (received by CAN Bus) can change counter value directly to avoid communication fault. (Not like mode0, in mode 0, it is clear counter when received watchdog value is different from previous one)
3 Work mode 2: RxWatchdog initial value is equal to period. RxWatchdog is free running down-counter each ms. When it arrive at 0, it will keep 0 and communication fault will occur. RxWatchdog Data packet (received by CAN Bus) can change counter value directly to avoid communication fault.

TxWatchdog has 4 kinds of work modes (TxWatchdog CAN Bus frame send out periodically

automatically by hardware for all modes):

1. Work mode 0: TxWatchdog value is decided by calling sendTxWatchdogValue node.
2. Work mode 1: TxWatchdog value increases 1 every TxWatchdog's period automatically. You don't need to call sendTxWatchdogValue node
3. Work mode 2: TxWatchdog value decreases 1 every TxWatchdog's period automatically. You don't need to call sendTxWatchdogValue node
4. Work mode 3: TxWatchdog will have no any value, its data packet length is zero. It is used for CANOpen Sync frame

Notes: *1. "Sync frame " can use "RxWatchdog/TxWatchdog frame" to replace, and in this situation, the CAN ID setting of sync frame will be ignored.*
2. All watchdogs are implemented in hardware. It is hardware real time. You can disable them and use Matlab/simulink to implement watchdog by yourself if you don't care hardware real time.

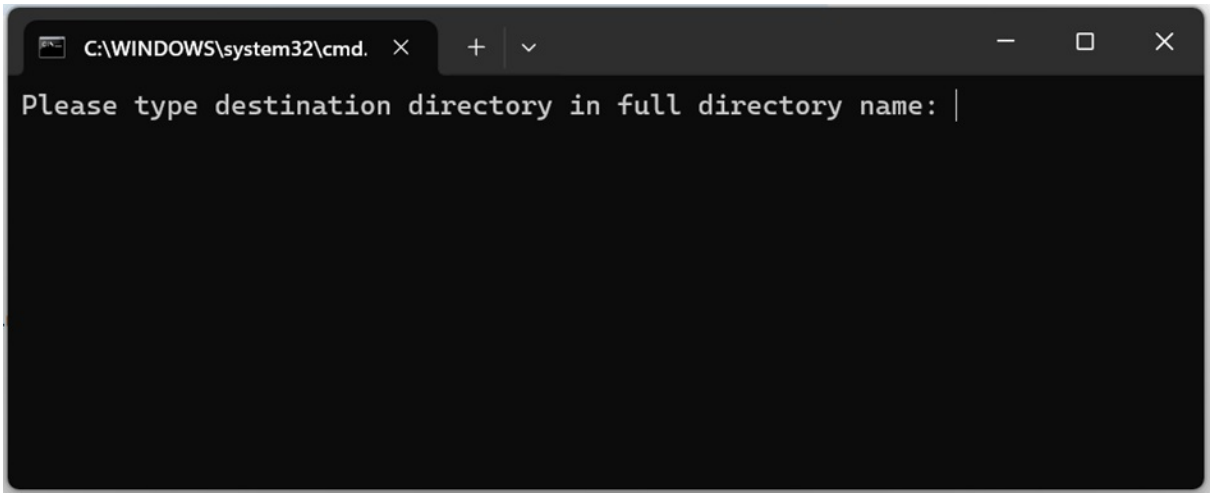
If you only use MATLAB, please only read "Matlab class for CAN bus controller".
If you only use Simulink, please only read "Simulink Model for CAN Bus Controller".

2 How to install CAN bus Controller in Matlab/Simulink?

Please follow steps below:

In Windows platform

- **Step1** Download CAN bus Controller library for Matlab/Simulink from clicking CANBusControl4Mat.zip
- **Step2** unzip CANBusControl4Mat.zip to any directory.
- **Step3** double click on setup.bat. It will display window below:

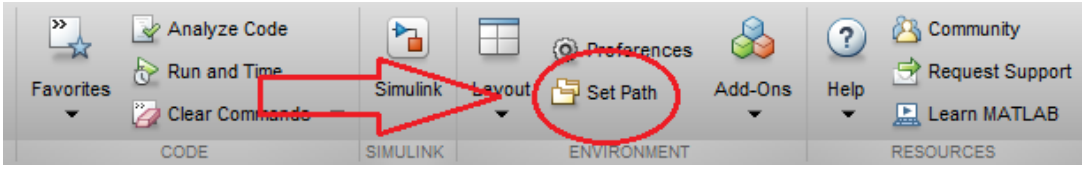


Following the instructions above cmd window, input your directory name you want to install, please use full directory including disk drive name such as "C:\myDir" (without quotation marks).

- **Step4** Set Matlab Path contains your destination directory of your CAN bus library.

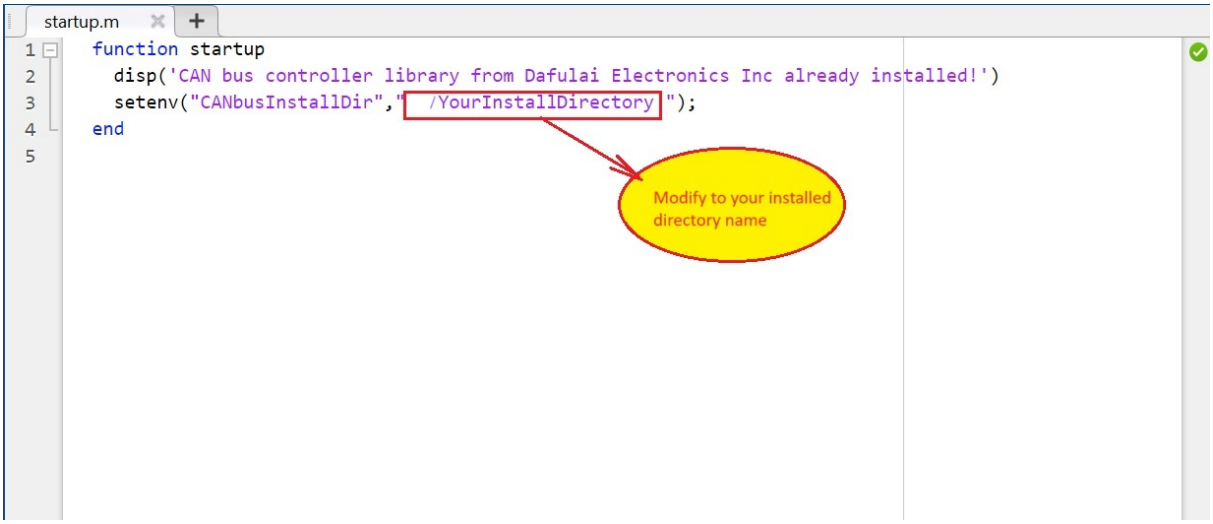
On Matlab's toolstrip, you may find the option "Set Path" which allows to select one directory

and save it permanently to Matlab's "search path". See screenshot below:



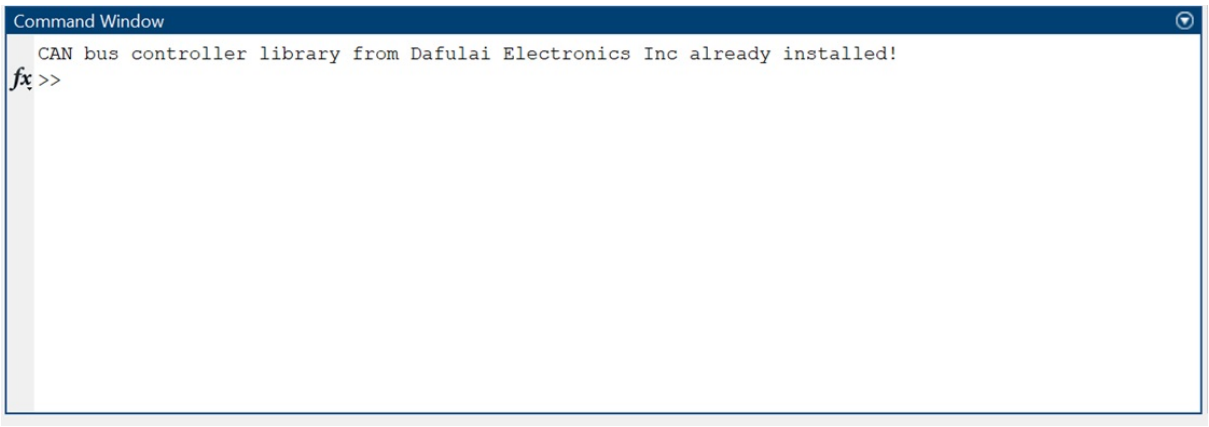
In non-Windows Platform.

- **Step1** Download CAN bus Controller library for Matlab/Simulink from clicking [CANBusControl4Mat.zip]
- **Step2** unzip CANBusControl4Mat.zip to directory you want to install.
- **Step3** Modify startup.m file. See screenshot below:

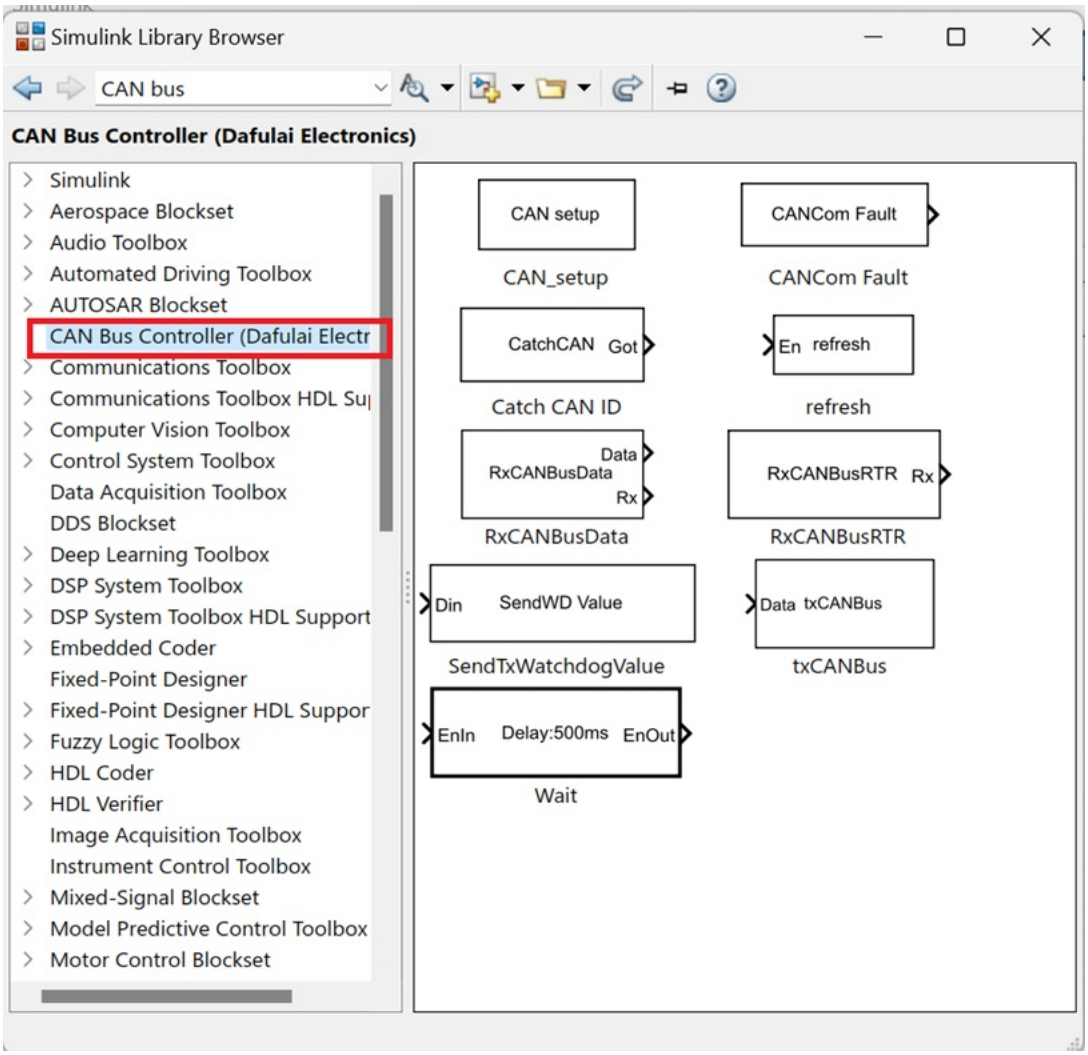


- **Step4** Set Matlab Path contains your directory you unzip. Now it is the same method as Step4 in windows platform.

After you finish CAN Bus library and Matlab Path setting correct, please re-start Matlab. You will see Message below in Matlab command window:



If you saw the above information, your CAN bus controller library install in your computer successfully. In your Simulink Library browser, you will see our CAN Bus controller library as shown as the following screenshot:



3 Matlab class for CAN bus controller

CANCtrl

Connection to CAN bus
Since R2019b

Description

A CANCtrl object represents a CAN BUS node for communication with the other CAN Bus nodes in the same CAN bus network. After creating the object, use dot notation to call methods.

Creation

Syntax

can = CANCtrl(portName)
can = CANCtrl(portName, Name, Value)

Description

example

can = CANCtrl(portName) connects to the CAN bus network by USB serial port specified by portName. It will use default CAN bus parameters for connecting.

can = CANCtrl(portName, Name, Value) connects to the CAN bus network by USB serial port specified by portName and sets additional private properties using optional name-value pair arguments.

Input Arguments

portName -- USB serial port name which is used by "CAN Bus controller" hardware.
character vector | string scalar

USB serial port name, specified as a character vector or string scalar. Use serialportlist to get a list of connected ports.

Example: "COM2"

Name-Value Arguments

Specify optional pairs of arguments as Name1=Value1,...,NameN=ValueN, where Name is the argument name and Value is the corresponding value. Name-value arguments must appear after portName, but the order of the pairs does not matter.

Please, use commas to separate each name and value, and enclose Name in quotes.

For example, can=CANCtrl("COM1","Sync", true,"SyncCANID", 38) will set Sync enabled and Sync

CAN ID = 38 (standard 11 bits of CAN ID)

You can use Name-Value pairs to set the following arguments:

- **Sync**

  Value is logical true or false. true means all trasnmittings will occur after special CAN ID frame (SyncCANID). It may be Transmitting CAN frame, it may be receiver CAN frame. It depends on "SyncByTransmit". Default=false

- **SyncCANID**

  Value is scalar number which denotes Sync CAN ID. if it is Extended frame, you must make Bit30 =1 (bit0 is MSb). Default=123

- **SyncPeriod**

  Value is scalar number which range from 1 to 65535. It denotes Sync periods in ms. Default=500

- **RxWatchdog**

  Value is logical true or false. true means Receiver Watchdog CAN Frame is enabled. Receiver Watchdog (We call it RxWatchdog) is for my controller to know whether CAN BUS communication fault. Default=false

- **TxWatchdog**

  Value is logical true or false. true means Transmitting Watchdog CAN Frame is enabled. Transmitting Watchdog (We call it TxWatchdog) is for other CAN Bus nodes to know whether CAN BUS communication fault. Default=false

- **SyncUseWatchdog**

  Value is logical true or false. true means we don't use SyncCANID for Sync. Instead of that, we use watchdog (maybe TxWatchdog, Maybe RxWatchdog. It depends on "SyncByTransmit") as Sync. Default=false

- **SyncByTransmit**

  Value is logical true or false. true means we use Transmitting CAN Frame as Sync. Default=false

- **RxWatchdogMode**

  Value is scalar number which can be 0, 1, 2. Default=0. It is RxWatchdog working mode. RxWatchdog has 3 different modes.

  Mode 0: Watchdog initial value is 0. Watchdog is free running up-counter each ms. Watchdog Counter will return 0 if we receive value from Watchdog CAN Bus frame, and value is different from previous received one.

  When Watchdog Counter is over RxWatchdog Period, it will keep the Rxwatchdog value and communication fault will occur. We don't need our Matlab to detect RxWatchdog. Hardware will do it automatically, and you just use public Property "Status" to get communication fault.

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.

RxWatchdog Data packet (receiving watchdog data packet) can change counter value to avoid communication fault. We don't need our Matlab to detect RxWatchdog. Hardware will do it automatically, and you just use public Property "Status" to get 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.

RxWatchdog Data packet (receiving watchdog data packet) can change counter value to avoid communication fault. We don't need our Matlab to detect RxWatchdog. Hardware will do it automatically, and you just use public Property "Status" to get communication fault.

- **RxWatchdogDLC**

  Value is scalar number which can be 1 to 8. Default=8. It is RxWatchdog Data frame's total length (Byte Qty). it is not RxWachdog data length.

- **TxWatchdogMode**

  Value is scalar number which can be 0, 1, 2, 3. Default=0. It is TxWatchdog working mode. TxWatchdog has 4 different modes.

  Mode 0 : TxWatchdog value is decided by Method "sendTxWatchdogValue". Value 's position and Length in CAN BUS data packet is decided by "TxWatchdogSartPos" and "TxWatchdogLen".

  You don't need to send CAN Bus TxWatchdog CAN Bus frame because Hardware sent out automatically,

  Mode 1: TxWatchdog value increases 1 every TxWatchdog's period by hardware automatically. You don't need to use Method "sendTxWatchdogValue" to give TxWatchdog value.

  Value 's position and Length in CAN BUS data packet is decided by "TxWatchdogSartPos" and "TxWatchdogLen". Value is unsigned. When value > maximum unsigned number, it will return 0.

  You don't need to send CAN Bus TxWatchdog CAN Bus frame because Hardware sent out automatically.

  Mode 2: TxWatchdog value decreases 1 every TxWatchdog's period by hardware automatically. You don't need to use Method "sendTxWatchdogValue" to give TxWatchdog value.

  Value 's position and Length in CAN BUS data packet is decided by "TxWatchdogSartPos" and "TxWatchdogLen". Value is unsigned. When value =0, it will return maximum unsigned number.

  You don't need to send CAN Bus TxWatchdog CAN Bus frame because Hardware sent out automatically.

Mode 3: TxWatchdog will have no any value, its data packet length is zero. It is used for CANOpen Sync frame.

- **RxWatchdogSartPos**

  Value is scalar number which can be 1 to 8. Default=1. It is RxWatchdog value's position (1 based) in frame's data field.

- **RxWatchdogLen**

  Value is scalar number 1 or 2. Default=2. It is RxWatchdog value' byte Qty.

- **TxWatchdogDLC**

  Value is scalar number which can be 1 to 8. Default=8. It is TxWatchdog Data frame's total length (Byte Qty). it is not TxWachdog data length.

- **TxWatchdogSartPos**

  Value is scalar number which can be 1 to 8. Default=1. It is TxWatchdog value's position (1 based) in frame's data field.

- **TxWatchdogLen**

  Value is scalar number 1 or 2. Default=1. It is TxWatchdog value' byte Qty.

- **RxWatchdogCANID**

  Value is scalar number which denotes RxWatchdog CAN ID. if it is Extended frame, you must make Bit30 =1 (bit0 is MSb). Default=234

- **RxWatchdogPeriod**

  Value is scalar number which range from 1 to 65535. It denotes RxWatchdog periods in ms. Default=500

- **TxWatchdogCANID**

  Value is scalar number which denotes TxWatchdog CAN ID. if it is Extended frame, you must make Bit30 =1 (bit0 is MSb). Default=345

- **TxWatchdogPeriod**

  Value is scalar number which range from 1 to 65535. It denotes TxWatchdog periods in ms. Default=500

- **Echo**

  Value is logical true or false. true means you can receive what you sent in CAN Bus. false means you cannot receive what you sent in CAN Bus. Default=false.

  If your CAN bus controller is "CAN Bus Analyzer only", your setting is no use, it will keep Echo= true even though you set it to false.

  Similarly, if your CAN bus controller is "CAN Bus Simulator only", your setting is no use, it will keep Echo= false even though you set it to true.

- **Terminator**

  Value is logical true or false. true means you will set hardware with 120 ohms CAN Bus terminator. Default=false

- **BaudRate**

  Value is scalar number which range is from 25000 (25K) to 1000000 (1M). Default=250000. It is CAN Bus baud rate.

  When its BaudRate<50000, 1500000/BaudRate must be integer. When BaudRate>=50000, 3000000/BaudRate must be integer.

- **Filters**

  Value is 16 numbers of vector which denotes 16 CAN Bus filters. 16 Filters are called F1, F2 ,..., F16. Our CAN bus controller has 3 Masks called M1, M2, M3.

  F1 to F8 are used M1, F9 to F12 are used M2, F13 to F16 are used M3. CAN bus Controller Hardware only receives CAN Frame with CANID meets the condition below:

  Fi "Logical bitwise And" M1 = CAN ID "Logical bitwise And" M1 (i=1 to 8) or
  Fi "Logical bitwise And" M2 = CAN ID "Logical bitwise And" M2 (i=9 to 12) or
  Fi "Logical bitwise And" M3 = CAN ID "Logical bitwise And" M3 (i=13 to 16)

  If you want to receive 29 bits of extended frame, you must keep filter and relative Mask both Bit30 =1 (bit0 is MSb), that means Fi=Fi + 2^30 and Mask=Mask+2^30.

  Of cause, if you want to receive 11 bits of standard frame, you must keep at least one filter and relative Mask both Bit30 =0 (bit0 is MSb)..

  Default=[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 2^30]. It means CAN bus controller receives all CAN bus frames if masks use default too. We propose customer to set up Filters and Masks when CAN BUS network has High traffic. Otherwise, don't touch it, just keep default.

- **Masks**

  Value is 3 numbers of vector which denotes 3 CAN Bus masks. Please read Filter above.

  Default=[0 0 2^30]. It means CAN bus controller receives all CAN bus frames if Filters use default too.

Examples

In general, you only need to set up USB serial port and CAN Bus baud rate, the default behaviour is disable all watchdogs and syncs. And receive all CAN bus frames (11 bits of standard CAN ID and 29 bits of extended CAN ID) except transmitted CAN Frames. The following statement is usually used in general project:

```
can=CANCtrl("COM2","baudrate", 250000);
```

In the following example, we will enable Sync, and it will only transmit CAN Frame only when CAN Bus controller receives standard frame with CAN ID=651:

```
can=CANCtrl("COM2","baudrate", 250000, "Sync", true, "SyncCANID", 651);
```

In the following example, we will enable Sync, and it will transmit extended frame with CAN ID=351 every 300ms automatically. Any transmit method only does truly transmitting after extended frame of CAN ID=351 transmitted.

```
can=CANCtrl("COM2","baudrate", 250000, "Sync", true, "SyncCANID", 351+2^30,
"SyncByTransmit", true, "SyncPeriod", 300);
```

In the following example, we will enable Sync. However, we use TxWatchdog to act as Sync. So you don't need to set Sync CAN ID and Sync period. TxWatchdog mode is 1, which means Watchdog counter increases 1 every 800ms. TxWatchdog CAN ID is extended 29 bits of 888. TxWatchdog Counter is 16 bits. and from CAN Bus frame data field's 1st byte start. So it will transmit extended frame with CAN ID=88 (DLC=4) every 800ms automatically. Any transmit method only does truly transmitting after extended frame of CAN ID=888 transmitted.

```
can=CANCtrl("COM2","baudrate", 250000, "Sync", true, "SyncUseWatchdog", true,
"SyncByTransmit", true,  ...
        "TxWatchdog" , true, "TxWatchdogDLC", 4, "TxWatchdogCANID", 888+2^30,
        "TxWatchdogMode", 1, ...
        "TxWatchdogSartPos", 1, "TxWatchdogLen", 2, "TxWatchdogPeriod", 800);
```

In the following example, firstly, we use "x" to denote 0 or 1 for an bit value (don't care). Our CAN BUS controller will receive standard CAN ID in binary below:

1. xxxxxxx1111 (use F1 and M1. So F1=00000001111 , M1=00000001111)
2. xxxxxxx0001 (use F2 and M1 So F2=00000000001 , M1=00000001111)
3. xxxxxxx0010 (use F3 and M1 So F3=00000000010 , M1=00000001111)
4. xxxxxxx0011 (use F4 to F8 and M1 So F4 to F8=00000000011 , M1=00000001111)

And Our CAN BUS controller will also receive extended CAN ID in binary below

1. x x111 1xxx xxxx xxxx xxxx xxxx xxxx (use F9 to F12 and M2 So F9 to F12 =0100 0111 1000 0000 0000 0000 0000 0000 , M2=0100 0111 1000 0000 0000 0000 0000 0000)
2. x x000 1xxx xxxx xxxx xxxx xxxx xxxx (use F13 to F16 and M3. So F13 to F16 =0100 0000 1000 0000 0000 0000 0000 0000 , M3=0100 0111 1000 0000 0000 0000 0000 0000)

The following statement will implement CAN bus object wuth above function:

```
can=CANCtrl("COM2","baudrate", 250000, "Filters", ...
      [15, 1, 2, 3, 3, 3, 3, 3, 1199570944, 1199570944, 1199570944, 1199570944, 1082130432,
1082130432, 1082130432, 1082130432], ...
      "Masks", [15, 1199570944, 1199570944]);
```

Properties

- Status — Read-Only. CAN bus network current status
  uint8 type row vector. Status(1) is CAN bus Rx Error count value, Status(2) is CAN bus Tx Error count value. Status(3) is BUS OFF (1: Bus off, 0: no bus off).
  Status(4) is CAN Bus communication fault flag based on RxWatchdog. (1 :Fault, 0: normal)

Object Functions (Or Methods)

transmitDataStandard	transmit 11 bits of standard CAN bus data frames with the same data type. Total data Byte qty<=8
transmitDataExtend	transmit 29 bits of extended CAN bus data frames with the same data type. Total data Byte qty<=8
transmitDataSingleFrame	transmit 11/29 bits of standard/extended CAN bus data single frame with different data type. Total data Byte qty<=8
transmitDataLongFrame	transmit 11/29 bits of standard/extended CAN bus data long frames with the same data type. Total data Byte qty>8
transmitRTRStandard	transmit 11 bits CAN bus remote request frames.
transmitRTRExtend	transmit 29 bits CAN bus remote request frames.
sendTxWatchdogValue	Send out TxWatchdog value. It is used when TxWatchdog Enable and mode is 0.
refresh	update CAN Bus receive buffers. If you want the latest data, you must call it.
receiveRaw	receive frames's raw data.
receiveSingleFrame	receive specified CAN ID single frame Data. (One frame with different data type in CAN Bus data field).
receiveMultipleFrames	receive specified CAN ID multiple frames Data (Multiple frames with the same CAN ID , for more than 8 bytes' data, and all data types are the same, The first byte of every frame is frame number.).
setCatchCANID	set up one special CAN ID we can catch.
enableCatchCANID	Enable/disable CAN ID catch function
setRxEventCallback	set up callback function for caught CAN ID

Events

SpecialCANCaught. After you use setCatchCANID method to set up a special CAN ID and use enableCatchCANID method to enable catch this CAN ID, when receive this CAN ID Frame, this event wull be created. If you already use setRxEventCallback method to set up your call back function, Matlab will execcute your call back function. In general, in call back function, you firstly call refresh method to update receiver buffers, and then use receive method to get data you want.

All methods details

transmitDataStandard
transmit 11 bits of standard CAN bus data frames with the same data type
Since R2019b

Syntax

transmitDataStandard(device,CANID, DLC, Data,DataType,Endian)

Description

Transmit 11 bits of standard CAN bus data frames It is used for transmitting data frames with the same data type. If different data type, please use method transmitDataSingleFrame

Input Arguments

- device — Connection to CAN bus
  CANCtrl object

- CANID — transmitted CAN Bus frame's CAN ID
  number vector for CAN ID of each standard data frame. This argument is mandatory.

- DLC — transmitted CAN Bus frame's DLC
  number vector for Data bytes Qty of each standard data frame. This argument is mandatory.

- Data — transmitted CAN Bus frame's Data fields
  number matrix for CAN bus Data. Row is frame No, column is data. This argument is mandatory.

- DataType — transmitted CAN Bus frame's Data type
  String scalar. It can be "uint8", "int8", "uint16","int16","uint32","int32","single","uint64","int64","double"
  Default="uint8"

- Endian — transmitted CAN Bus frame's Data Endian.
  String scalar. It can be "Little-Endian", "Big-Endian".
  Default="Little-Endian"

Examples

We send standard CAN Bus data, Data type is uint16, and little-endian. Every frame has 4 uint16 data (8 bytes).
The first frame : CAN ID= 12, DLC=8, and 4 uint16 data are "367, 789, 8901, 9992"
The second frame: CAN ID=67, DLC=8, and 4 uint16 data are "1367, 2000, 3000, 8001"

Firstly, we set up a CANCtrl object with COM2 and CAN Bus baud rate: 250k:

```
can=CANCtrl("COM2","baudrate", 250000);
```

Then we prepare input arguments:

```
CANID=[12 67];
DLC=[8 8];
Data=[367 789 8901 9992; 1367 2000 3000 8001];
```

At last, we call CANCtrl object's method to transmit CAN frames:

```
transmitDataStandard(can,CANID,DLC.Data,"uint16","Little-Endian");
```

transmitDataExtend

transmit 29 bits of extended CAN bus data frames with the same data type
Since R2019b

Syntax

transmitDataExtend(device,CANID, DLC, Data,DataType,Endian)

Description

Transmit 29 bits of extended CAN bus data frames It is used for transmitting data frames with the same data type. If different data type, please use method transmitDataSingleFrame

Input Arguments

- device — Connection to CAN bus
  CANCtrl object

- CANID — transmitted CAN Bus frame's CAN ID
  number vector for CAN ID of each standard data frame. This argument is mandatory.

- DLC — transmitted CAN Bus frame's DLC
  number vector for Data bytes Qty of each standard data frame. This argument is mandatory.

- Data — transmitted CAN Bus frame's Data fields
  number matrix for CAN bus Data. Row is frame No, column is data. This argument is mandatory.

- DataType — transmitted CAN Bus frame's Data type
  String scalar. It can be "uint8", "int8", "uint16","int16","uint32","int32","single","uint64","int64","double"
  Default="uint8"

- Endian — transmitted CAN Bus frame's Data Endian.
  String scalar. It can be "Little-Endian", "Big-Endian".
  Default="Little-Endian"

Examples

We send extended CAN Bus data, Data type is int32, and big-endian.
The first frame : CAN ID= 8412, DLC=8, and 2 int32 data are "-32189, 98765"
The second frame: CAN ID=9867, DLC=4, and only one int32 data are "-999998"

Firstly, we set up a CANCtrl object with COM2 and CAN Bus baud rate: 125k:

```
can=CANCtrl("COM2","baudrate", 125000);
```

Then we prepare input arguments:

```
CANID=[8412 9867];
DLC=[8 4];
Data=[-32189  98765;  -999998  0];
```

At last, we call CANCtrl object's method to transmit CAN frames:

```
transmitDataExtend(can,CANID,DLC.Data,"int32","Big-Endian");
```

transmitDataSingleFrame

transmit 11/29 bits of standard/extended CAN bus data single frame with the different data type
Since R2019b

Syntax

transmitDataSingleFrame(device, CANID, Extended, Data, DataType, Endian)

Description

Transmit 11/29 bits of standard/extended CAN bus data single frame. It is used for transmitting data frames with the different data type. If the same data type, please use method transmitDataStandard or transmitDataExtend

Input Arguments

- device — Connection to CAN bus
  CANCtrl object
- CANID — transmitted CAN Bus frame's CAN ID
  number scalar for CAN ID. This argument is mandatory..
- Extended — transmitted CAN Bus extended frame or standard frame
  logical scalar. true means extended data frame. This argument is mandatory.
- Data — transmitted CAN Bus frame's Data fields
  number vector for CAN bus Data in single frame. This argument is mandatory.
- DataType — transmitted CAN Bus frame's Data type
  String vector. It denotes each data type in CAN Bus Data field.. It can be "uint8", "int8", "uint16","int16","uint32","int32","single","uint64","int64","double"
  Default="uint8"
- Endian — transmitted CAN Bus frame's Data Endian.
  String scalar. It denotes each data type in CAN Bus Data field. It can be "Little-Endian", "Big-Endian".
  Default="Little-Endian"

Examples

We send extended CAN Bus data, 3 data, the 1st Data is uint8, .value is 125, the 2nd data is int16, value is -12560, the 3rd data is single, value is 412.38. We use "Big-Endian" for int16 and single CAN ID is 59904. We have to make sure total byte qty<=8. In this example, total byte qty=1+2+4=7. It is OK. And method will use DLC=7 automatically.

Firstly, we set up a CANCtrl object with COM2 and CAN Bus baud rate: 125k:

```
can=CANCtrl("COM2","baudrate", 125000);
```

Then we prepare input arguments:

```
CANID=59904;
Data=[125 -12560  412.38];
DataType=["uint8"  "int16"  "single"];
```

At last, we call CANCtrl object's method to transmit CAN frames:

```
transmitDataSingleFrame(can,CANID, true. Data, DataType ,"Big-Endian");
```

transmitDataLongFrame

transmit 11/29 bits of standard/extended CAN bus data long frames with the same data type
Since R2019b

Syntax

transmitDataLongFrame(device, CANID, Extended, Data, DataType, Endian, FrameNoStart, isLastFrameDLC8)

Description

Transmit 11/29 bits of standard/extended CAN bus data long frames. Long frames means that multiple frames with the same CAN ID and first data byte is frame number starting from 0 or 1 in general. The other 7 bytes in long frames data field will be truly data. Every frame number from long frame is increased 1 consecutively. The last frame's DLC can be 8 or less. This method is used for transmitting data frames with the same data type. but data bytes qty is over 8. The first byte in each frame is frame number.

Input Arguments

- device — Connection to CAN bus
  CANCtrl object
- CANID — transmitted CAN Bus frame's CAN ID
  number scalar for CAN ID. This argument is mandatory..
- Extended — transmitted CAN Bus extended frame or standard frame
  logical scalar. true means extended data frame. This argument is mandatory.
- Data — transmitted CAN Bus frame's Data fields
  number vector for CAN bus Data in single frame. This argument is mandatory.
- DataType — transmitted CAN Bus frame's Data type
  String scalar. It denotes each data type in CAN Bus Data field.. It can be "uint8", "int8", "uint16","int16","uint32","int32","single","uint64","int64","double"
  Default="uint8"
- Endian — transmitted CAN Bus frame's Data Endian.
  String scalar. It denotes each data type in CAN Bus Data field. It can be "Little-Endian", "Big-Endian".
  Default="Little-Endian"
- FrameNoStart
  number scalar. It denotes the first frame's frme number. Default=0.
- isLastFrameDLC8
  logical scalar. true means that the DLC of the last frame is 8. the remainder data of last frame will be zero. false means that system will use actual DLC for the last frame.
  Default= false

Examples

We send standard CAN Bus data, 12 data, Data type is uint16, .value is 125, 89, 678, 134, 789, 444, 555, 666,777,111,222, and 888. We use "Big-Endian" for uint16
CAN ID is 599. FrameNoStart=3. The last frame DLC uses actual length we need.

Firstly, we set up a CANCtrl object with COM2 and CAN Bus baud rate: 125k:

```
can=CANCtrl("COM2","baudrate", 125000);
```

Then we prepare input arguments:

```
CANID=599;
Data=[125, 89, 678, 134, 789, 444, 555, 666,777,111,222, 888];
DataType="uint16";
```

At last, we call CANCtrl object's method to transmit CAN frames:

```
transmitDataLongFrame(can,CANID, false. Data, DataType ,"Big-Endian", 3, false);
```

transmitRTRStandard

transmit 11 bits CAN bus remote request frames.
Since R2019b

Syntax

transmitRTRStandard(device,CANID)

Description

Transmit 11 bits of standard CAN bus remote request frames It is used for transmitting multiple RTR frames with different CAN ID.

Input Arguments

- device — Connection to CAN bus
  CANCtrl object
- CANID — transmitted CAN Bus frame's CAN ID
  number vector for CAN ID of each standard RTR frame. This argument is mandatory.

Examples

We send 3 standard CAN Bus remote request frames.
The first frame : CAN ID= 12.
The second frame: CAN ID=67.
The third frame: CAN ID=84.

Firstly, we set up a CANCtrl object with COM2 and CAN Bus baud rate: 250k:

```
can=CANCtrl("COM2","baudrate", 250000);
```

Then we prepare input arguments:

```
CANID=[12 67 84];
```

At last, we call CANCtrl object's method to transmit CAN frames:

```
transmitRTRStandard(can,CANID);
```

transmitRTRExtend

transmit 29 bits CAN bus remote request frames.
Since R2019b

Syntax

transmitRTRExtend(device,CANID)

Description

Transmit 29 bits of extended CAN bus remote request frames It is used for transmitting multiple RTR frames with different CAN ID.

Input Arguments

- device — Connection to CAN bus
  CANCtrl object
- CANID — transmitted CAN Bus frame's CAN ID
  number vector for CAN ID of each extended RTR frame. This argument is mandatory.

Examples

We send 3 extended CAN Bus remote request frames.
The first frame : CAN ID= 12345.
The second frame: CAN ID=67000.
The third frame: CAN ID=84989.

Firstly, we set up a CANCtrl object with COM2 and CAN Bus baud rate: 250k:

```
can=CANCtrl("COM2","baudrate", 250000);
```

Then we prepare input arguments:

```
CANID=[12345 67000 84989];
```

At last, we call CANCtrl object's method to transmit CAN frames:

```
transmitRTRExtend(can,CANID);
```

sendTxWatchdogValue

change Tx Watchdog counter value..
Since R2019b

Syntax

sendTxWatchdogValue(device, TxWDValue)

Description

Send out TxWatchdog value. It is used when TxWatchdog Enable and mode is 0..

Input Arguments

- device — Connection to CAN bus
  CANCtrl object

- TxWDValue — TxWatchdog counter value
  number scalar for TxWatchdog counter value. This argument is mandatory.

Examples

We enabled TxWatchdog. And its work mode is 0. TxWatchdog Periods=300ms, TxWatchdog frame has 8 bytes. However, TxWatchdog counter has 2 bytes and start from the first byte of TxWatchdog CAN Bus frame.
We will send TxWatchdog counter value every 600ms. and Counter value increase 5 every 600ms. If it over maximum value of 2 bytes (65535), it will be back to zero.

Firstly, we set up a CANCtrl object with COM2 and CAN Bus baud rate: 250k , and set up TxWatchdog as we expected.

```
can=CANCtrl("COM2","baudrate", 250000, "TxWatchdog", true,  "TxWatchdogCANID", 56,  ...
                "TxWatchdogDLC", 8, "TxWatchdogSartPos", 1, "TxWatchdogLen" , 2,
                "TxWatchdogMode", 0 , "TxWatchdogPeriod", 300);
```

And we set up counter initial value=0

```
counter=0;
```

then, we call CANCtrl object's method to send out counter value every 600ms

```
while true
    sendTxWatchdogValue(can, counter);
    counter=counter+5;    % modify counter value
    if counter>65535
        counter=0;
    end
    pause(0.6)      % pause 600 ms for sending every 600ms
end
```

refresh

update CAN Bus receive buffers. If you want the latest data, you must call it.
Since R2019b

Syntax

refresh(device,)

Description

let CAN bus receiver buffers update to dictate the latest receiver frames. After executing this method, all old buffers will disappear, and buffers will be filled with all receiver frames from previous refresh to this time's refresh.
If you want the latest frame, you should call it before you call any receive method, otherwise, you still get old data.

Input Arguments

- device — Connection to CAN bus

CANCtrl object

receiveRaw

receive CAN Bus frames's raw data.
Since R2019b

Syntax

[CANID,Transmit, Extended, RTR, DLC, Data, TimeStamp,Status] = receiveRaw(device)

Description

get CAN Bus received frames's raw data. You must call refresh method before calling it in order to get new frame data. You can know whether received frame is self-transmitted or RTR, and you can know time stamp and Communication status in additional to received data byte.
This method is non-block function. If no any frame received, it will get empty vector for CAN ID.

Input Arguments

- device — Connection to CAN bus
  CANCtrl object

Output Arguments

- CANID — received frames' CAN ID
  uint32 row vector for received CAN ID.
- Transmit — received CAN frame is my transmitted frame
  logical row vector. true means received frame is what it transmitted frame.
- Extended — received CAN frame is 29 bits' extended frame
  logical row vector. true means received frame is 29 bits of extended frame.
- RTR — received CAN frame is remote request frame
  logical row vector. true means received frame is remote request frame.
- DLC — received frames' DLC
  uint8 row vector which tells you DLC of received frame.
- Data — received frames' Data field
  uint8 Matrix, which denotes received frame's data. row is frame number, column is data number. Matrix has 8 columns which means maximum 8 bytes.
- TimeStamp — received CAN frames' time stamp in seconds
  double type row vector. it denotes each frame's received time stamp in second. Time start point (0 value point) is from time you create CAN BUS object.
- Status — CAN bus network current status
  uint8 type row vector. Status(1) is CAN bus Rx Error count value, Status(2) is CAN bus Tx Error count value. Status(3) is BUS OFF (1: Bus off, 0: no bus off).
  Status(4) is CAN Bus communication fault flag based on RxWatchdog. (1 :Fault, 0: normal)

Examples

The following example will receive CAN bus frames every 200ms.

Firstly, we set up a CANCtrl object with COM2 and CAN Bus baud rate: 250k.

```
can=CANCtrl("COM2","baudrate", 250000 );
```

then, we call CANCtrl object's method refresh and receiveRaw to get received frame information every 200ms

```
while true
    pause(0.2)      % pause 200 ms for receiving every 200ms
    refresh( can );
    [CANID,Transmit, Extended, RTR, DLC, Data, TimeStamp,Status] = receiveRaw( can )
end
```

receiveSingleFrame

receive specified CAN ID single frame Data with different data type.
Since R2019b

Syntax

[DataOut, Exist]=receiveSingleFrame(device, CANIDTarget, ExtendedTarget, RTRTarget, DataType, Endian)

Description

This is non-block method. It will receive specified CAN ID single frame Data. (One frame with different data type in CAN Bus data field). If not received, Exist will be false. You must call refresh method before call it in order to get the new data.

Input Arguments

- device — Connection to CAN bus
  CANCtrl object
- CANIDTarget — your interested received frames' CAN ID
  number type scalar for received CAN ID. This is mandatory argument.
- ExtendedTarget — you are interested in extended CAN ID
  logical scalar. true means we only need extended frame. false means we only need standard frame. This is mandatory argument.
- RTRTarget — you are interested in remote request frame.
  logical scalar. true means we only need remote request frame. false means we only need data frame. This is mandatory argument.
- DataType — Data type for your received CAN Frame.
  String vector. It denotes each data type in CAN Bus Data field. It can be "uint8", "int8", "uint16","int16","uint32","int32","single","uint64","int64","double"
  Default="uint8".

- Endian — the order of multiple bytes data.
  String scalar. It denotes CAN Bus Data frame's Data Endian. It can be "Little-Endian", "Big-Endian". Default="Little-Endian"

Output Arguments

- DataOut — received frames' Data
  double row vector. It denotes each data we received from specified CAN ID. For RTR frame or no any frame received, DataOut is empty.

- Exist — whether received the frame we are interested in
  logical scalar. true means we got what we expected CAN bus frame.

Examples

The following example will receive CAN bus frames every 200ms.
We expect CAN ID=12 standard frame with 1 data and int32 data type, and 1 data with single data type.
We expect CAN ID=12 extended frame with 3 data and uint16 data type. All data are "Little-Endian"

Note: Although these 2 CAN Bus frames' CAN ID is 12, they are different CAN ID because one is 11 bits' standard CAN ID, the other is 29 bits' extended CAN ID

Firstly, we set up a CANCtrl object with COM2 and CAN Bus baud rate: 250k.

```
can=CANCtrl("COM2","baudrate", 250000 );
```

then, we call CANCtrl object's method refresh and receiveSingleFrame to get data we expected every 200ms

```
while true
    pause(0.2)      % pause 200 ms for receiving every 200ms
    refresh( can );    % refresh receiver buffers
    [DataOut1, Exist1]=receiveSingleFrame( can, 12, false, false, ["int32"  "single"]);
    if Exist1
        disp("standard frame Data:")
        disp(DataOut1)
    end
    [DataOut2, Exist2]=receiveSingleFrame( can, 12, true, false, ["int16"  "int16"  "int16" ]);
    if Exist2
        disp("extended frame Data:")
        disp(DataOut2)
    end
end
```

receiveMultipleFrames

receive specified CAN ID multiple frames Data with the same data type.
Since R2019b

Syntax

[DataOut, Exist]=receiveMultipleFrames (device, CANIDTarget, ExtendedTarget, DataQty, DataType, Endian, FrameNoStart)

Description

This is non-block method. It will receive specified CAN ID multiple frames Data (Multiple frames with the same CAN ID , and more than 8 bytes' data, and all data types are the same , The first byte of every frame is frame number.). If not received, Exist will be false. You must call refresh method before call it in order to get the new data. And you must make sure all frames are ready before you call refresh method. You can use Call back function of event "SpecialCANCaught" to guarantee.

Input Arguments

- device — Connection to CAN bus
  CANCtrl object

- CANIDTarget — your interested received frames' CAN ID
  number type scalar for received CAN ID. This is mandatory argument.

- ExtendedTarget — you are interested in extended CAN ID
  logical scalar. true means we only need extended frame. false means we only need standard frame. This is mandatory argument.

- DataQty — Total data items for all frames received..
  number type scalar. It denotes how many data items we will receive (data unit is DatatType parameter below) . This is mandatory argument..

- DataType — Data type for your received CAN Frame.
  String scalar. It denotes data type in CAN Bus Data field. It can be "uint8", "int8", "uint16","int16","uint32","int32","single","uint64","int64","double"
  Default="uint8".

- Endian — the order of multiple bytes data.
  String scalar. It denotes CAN Bus Data frame's Data Endian. It can be "Little-Endian", "Big-Endian".
  Default="Little-Endian"

- FrameNoStart — the frame number of the first frame..
  number scalar. It denote the first frame's frame number. The value is 0 to 255. Default=0.

Output Arguments

- DataOut — received frames' Data
  double row vector. It denotes all data we received from multiple frames. If long data frames are not fully ready (received entire long frames in the frame Number order), DataOut will be empty and Exist=false.

- Exist — whether received the frame we are interested in
  logical scalar. true means we got what we expected CAN bus frame.

Examples

The following example will receive CAN bus frames every 200ms.
We expect CAN ID=12 standard frame with 43 data with int16 data type and all data are "Little-Endian". The first frame number is 0

Firstly, we set up a CANCtrl object with COM2 and CAN Bus baud rate: 250k.

```
can=CANCtrl("COM2","baudrate", 250000 );
```

then, we call CANCtrl object's method refresh and *receiveMultipleFrames* to get data we expected every 200ms

```
while true
    pause(0.2)      % pause 200 ms for receiving every 200ms
    refresh( can );    % refresh receiver buffers
    [DataOut, Exist]=receiveMultipleFrames (can, 12, false, 43, "uint16");
    if Exist
        disp("standard frame Data:")
        disp(DataOut)
    end
end
```

The above example is not perfect. If the time of refresh is before receiving the last frame, it will have problem. You can set one Flag variable to true in the Call back function of event "SpecialCANCaught" (catch the last frame of long frame). For example, Flag variable is "Got_CANID". The example code is modified as below:

```
while true
    if Got_CANID
        pause(0.2)      % pause 200 ms for receiving every 200ms
        Got_CANID=false;
        refresh( can );    % refresh receiver buffers
        [DataOut, Exist]=receiveMultipleFrames (can, 12, false, 43, "uint16");
        if Exist
            disp("standard frame Data:")
            disp(DataOut)
        end
    end
end
```

setCatchCANID

set up one special CAN ID we can catch
Since R2019b

Syntax

setCatchCANID(device, CANID, Extended, CatchSpecialframe, SpecialFrameNo)

Description

It will set up a special CAN ID. When this special CAN ID received, CAN BUS controller object will create event "SpecialCANCaught" if caught CAN ID enabled and CatchSpecialframe is false. Or when this special CAN ID received and the first byte of data field=SpecialFrameNo, we will create event "SpecialCANcaught" if caught CAN ID is enabled and CatchSpecialframe is true. Event "SpecialCANcaught" will call "callback function". Callback function is customized function, You can use method "setRxEventCallback" to set it.

Input Arguments

- device — Connection to CAN bus
  CANCtrl object
- CANID — you want to catch CAN ID
  number type scalar for caught CAN ID. This is mandatory argument.
- Extended — you want to catch CAN ID
  logical type scalar for CAN ID type of caught CAN frame. Default=false, means 11 bits of standard frame.
- CatchSpecialframe — Catch special data field with the 1st byte=SpecialFrameNo
  logical type scalar. True means we will compare 1st byte of caught CAN frame besides CAN ID and Extended, we only create event "SpecialCANcaught" when both CAN ID / Extended and first bye match. False means we only care CAN ID and Extended value , we don't care RTR and the 1st byte. Default=false.
- SpecialFrameNo — the 1st byte value of caught CAN bus data frame
  number type scalar for the 1st byte value of CAN bus data field. Data range is 0 to 255. Default=2.

Examples

The following example will set up caught CAN ID=standard 11 bits' 456. Don't care data field.

Firstly, we set up a CANCtrl object with COM2 and CAN Bus baud rate: 250k.

```
can=CANCtrl("COM2","baudrate", 250000 );
```

then we use the following statement to set up caught CAN ID:

```
setCatchCANID( can, 456, false );
```

enableCatchCANID

Enable/disable CAN ID catch function
Since R2019b

Syntax

enableCatchCANID(device, enable)

Description

It will enable/disable "Caught CAN ID". And if it is enabled and "Caught CAN ID" received and also other conditions are met, it will create event "SpecialCANCaught". It leads to callback function executing.

Input Arguments

- device — Connection to CAN bus
  CANCtrl object
- enable — True will enable.

logical scalar. true means we enable "Catch CAN ID". Default=true..

Examples

The following example will set up caught CAN ID=standard 11 bits' 456. and Enable it.

Firstly, we set up a CANCtrl object with COM2 and CAN Bus baud rate: 250k.

```
can=CANCtrl("COM2","baudrate", 250000 );
```

then we use the following statement to set up caught CAN ID:

```
setCatchCANID( can, 456, false );
```

At last, we enable it:

```
enableCatchCANID( can );
```

setRxEventCallback

set up callback function for caught CAN ID
Since R2019b

Syntax

setRxEventCallback(device, myCallback)

Description

It will set up callback function for event "SpecialCANCaught".

Input Arguments

- device — Connection to CAN bus
  CANCtrl object
- myCallback — callback function name
  Customer's function. Function prototype is " FunctionName(src,event) "
  Inside call back function, you will call src.fresh and src's receiveRaw method or src's receiveSingleFrame or receiveMultipleFrames methods to get received data.

Examples

The following example will set up caught CAN ID=standard 11 bits' 456. and Enable it. And when CAN ID 456 Coming, call back function will be called.

Firstly, we set up a CANCtrl object with COM2 and CAN Bus baud rate: 250k.

```
can=CANCtrl("COM2","baudrate", 250000 );
```

then we use the following statement to set up caught CAN ID:

setCatchCANID(can, 456, false);

We write a call back function:

myCallback=@(src,evt) (disp("ok"));

And we use setRxEventCallback to set up my Call back function:

setRxEventCallback(can, myCallback)

At last, we enable "Caught CAN ID":

enableCatchCANID(can);

When CAN Bus controller receives standard CANID=456, it will call myCallback, so "ok" will be displayed.

4 Simulink Model for CAN Bus Controller

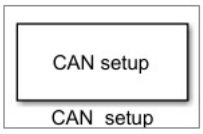
We will introduce every block diagram node for CAN Bus controller in the following sections.

4.1 CAN_setup

CAN_setup

Set up CAN Bus controller (CAN Bus Analyzer/Simulator from Dafulai Electronic Inc)
Since R2019b

Library: CAN Bus Controller (Dafulai Electronics) /CAN_setup



Description

This block sets up all parameters of CAN Bus Analyzer/Simulator from Dafulai Electronic Inc
And it will refresh all Can bus receiver buffers every sampling cycle automatically if parameter "Enable auto-refresh" is true (Default is true). Otherwise, you must call "refresh" block by yourself in order to get the latest received frames.
You must put this block in your simulation model in order to access CAN Bus by CAN Bus Analyzer/ Simulator.

Firstly, let us introduce CAN Bus Controller function.

Our CAN Bus controller can transmit/Receive standard or/and extended CAN Bus data and RTR frames as general CAN bus node.

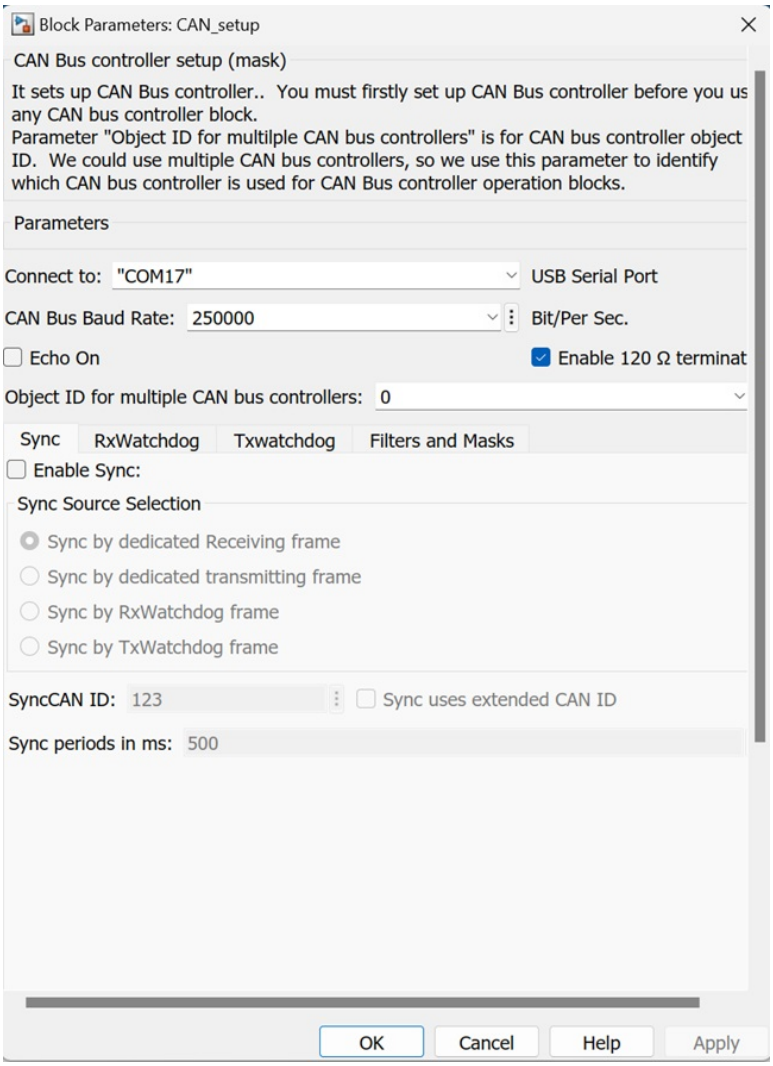
And furthermore, transmitting CAN Bus Frame can be synchronized by special CAN BUS transmitting frame or receiving frame. If Sync is enabled, CAN Bus controller only can transmit CAN Bus frame when it receives or transmits this special CAN BUS frame. This special CAN BUS frame is called

"Sync frame"
Another feature is that our CAN Bus Controller has Watchdog function.
Watchdog purpose is for getting communication fault information.
Received Watchdog (Call it RxWatchdog) is for itself to know whether CAN BUS communication OK or not.
Transmitted Watchdog (Call it TxWatchdog) is for other CAN bus nodes to know whether CAN BUS communication OK or not.
TxWatchdog CAN Bus frame send out periodically automatically by hardware, You don't need to send by calling transmit block.
Similarly, if "Sync frame" is "Transmit frame", "Sync frame" send out periodically automatically by hardware, You don't need to send by calling transmit block.

This block will set up "Sync" frame and TxWatchdog/RxWatchdog frame, and set up Filter/mask in order to decrease receiving traffic from hardware.

Parameters

There are many parameters for this block. Please double click this block to open parameters dialog below:



Many parameters are self-explanation from the label. We only explain some special parameters.

- Echo On — true means we can receive the CAN bus frame which is we transmit out. It is useful for CAN bus controller which supports both Analyzer and Simulator functions. If it only supports Analyzer, Echo On will actually keep true no matter which value is set. Similarly, If it only supports Simulator, Echo On will actually keep false no matter which value is set.

- Object ID for multiple CAN Bus controllers — In one PC, we may use multiple CAN bus controllers, this is for identifying each one.

- Sync Source Selection — Sync may from one of 4 different source.
  When source is from RxWatchdog or TxWatchdog. You will use RxWatchdog or TxWatchdog CAN ID, just ignore SyncCANID parameter. In this situation, Watchdog will have dual functions, one is for watchdog, the other is for Sync.

- RxWatchdog Mode — RxWatchdog has 3 different modes (You must to click tab "RxWatchdog" to show up)

  Mode 0: Watchdog initial value is 0. Watchdog is free running up-counter each ms.

Watchdog Counter will return 0 if we receive value from Watchdog CAN Bus frame, and value is different from previous received one.

When Watchdog Counter is over RxWatchdog Period, it will keep the Rxwatchdog value and communication fault will occur. We don't need our Matlab to detect RxWatchdog. Hardware will do it automatically, and you just use refresh method and receiveRaw method to get status.

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.

RxWatchdog Data packet (receiving watchdog data packet) can change counter value to avoid communication fault. We don't need our Matlab to detect RxWatchdog. Hardware will do it automatically, and

you just use refresh method and receiveRaw method to get status.

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.

RxWatchdog Data packet (receiving watchdog data packet) can change counter value to avoid communication fault. We don't need our Matlab to detect RxWatchdog. Hardware will do it automatically, and

you just use refresh method and receiveRaw method to get status.

- TxWatchdog Mode — TxWatchdog has 4 different modes (You must to click tab "TxWatchdog" to show up)

Mode 0 : TxWatchdog value is decided by Method "sendTxWatchdogValue". Value 's position and Length in CAN BUS data packet is decided by "TxWatchdogSartPos" and "TxWatchdogLen".

You don't need to send CAN Bus TxWatchdog CAN Bus frame because Hardware sent out automatically,

Mode 1: TxWatchdog value increases 1 every TxWatchdog's period by hardware automatically. You don't need to use Method "sendTxWatchdogValue" to give TxWatchdog value.

Value 's position and Length in CAN BUS data packet is decided by "TxWatchdogSartPos" and "TxWatchdogLen". Value is unsigned. When value > maximum unsigned number, it will return 0.

You don't need to send CAN Bus TxWatchdog CAN Bus frame because Hardware sent out automatically.

Mode 2: TxWatchdog value decreases 1 every TxWatchdog's period by hardware automatically. You don't need to use Method "sendTxWatchdogValue" to give TxWatchdog value.

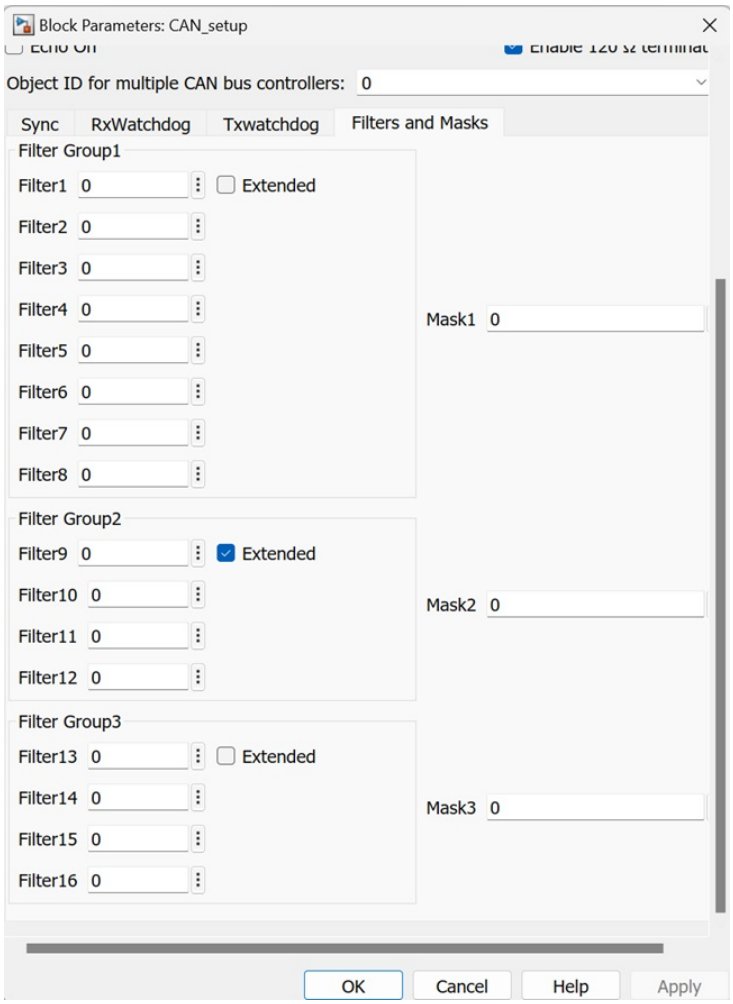
Value 's position and Length in CAN BUS data packet is decided by "TxWatchdogSartPos" and "TxWatchdogLen". Value is unsigned. When value =0, it

will return maximum unsigned number.

You don't need to send CAN Bus TxWatchdog CAN Bus frame because Hardware sent out automatically.

Mode 3: TxWatchdog will have no any value, its data packet length is zero. It is used for CANOpen Sync frame.

- Filter and Mask — 16 Filters and 3 Masks (You must to click tab "Filters and Masks" to show up)



Filter1 to Filter8 (Group1) are used Mask1, Filter9 to Filter12 (Group2) are used Mask2, Filter13 to Filter16 (Group3) are used Mask3. CAN bus Controller Hardware only receives CAN Frame with CAN ID meets the condition below:

FilterN "Logical bitwise And" Mask1 = CAN ID "Logical bitwise And" Mask1 (N=1 to 8) or

FilterN "Logical bitwise And" Mask2 = CAN ID "Logical bitwise And" Mask2 (N=9 to 12) or

FilterN "Logical bitwise And" Mask3 = CAN ID "Logical bitwise And" Mask3 (N=13 to 16)

That is to say, If bit in Mask =1, we will care the same bit in received CAN ID, it must be the bit value in filter. Otherwise, if bit in Mask =0, we can receive any bit value in received CAN ID.

- Enable auto-refresh — Enable refresh received buffer automatically every sampling time. Default is true. It means that model will update CAN Bus received frame buffers automatically every sampling time. You don't need to put "refresh" block into your simulink model. This parameter is on the bottom of parameter dialog, you may need scroll down to see it.

Ports

Input

None

Outpot

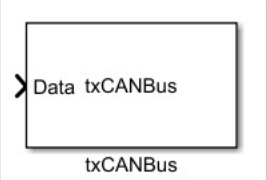
None

4.2 txCANBus

txCANBus

transmit CAN Bus frame
Since R2019b

Library: CAN Bus Controller (Dafulai Electronics) /txCANBus



Description

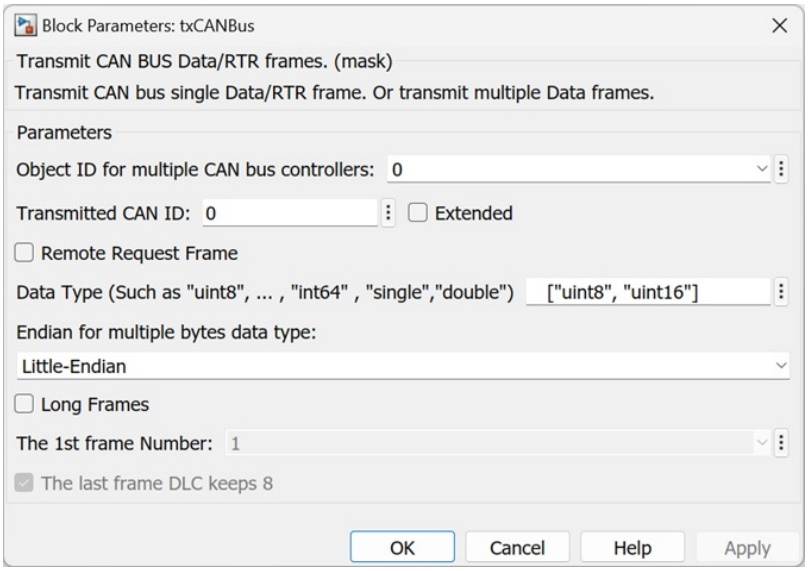
This block transmits single CAN Data/RTR frame, and long Data frames.

For single Data frame, DLC is decided by Data Input vector.
For long Data frames, frame Qty is decided by Data Input vector.

If "Sync" is disabled in CAN_setup block, this frame (these frames) will be transmitted immediately. Otherwise, it will be transmitted after "Sync" frame.

Parameters

Please double click this block to open parameters dialog below:



Let us explain parameters.

- Object ID for multiple CAN Bus controllers — In one PC, we may use multiple CAN bus controllers, this is for identifying each one we use. It must match the same parameter in CAN_setup block.
- Transmitted CAN ID — transmitted CAN bus frame's CAN ID. It is scalar.
- Extended — true means "Transmitted CAN ID" is 29 bits of extended. Otherwise, it is 11 bits of standard.
- Remote Request Frame — true means "Transmit RTR frame". Otherwise, it means "Transmit Data frame".
- Data Type — Input Port vector's Target Data type. For single data frame, It is vector. The element of this vector is string. The valid string is "uint8", "uint16", "uint32", "uint64", "int8", "int16", "int32", "int64", "single", "double". It denotes "Target data type in frame data field" for each data element in input port "Data" although all elements' data type in "Data" vector is "double". For long data frames, it is scalar string because all target data type is the same for long frames.
- Endian for multiple bytes data type — Big-Endian or Little-Endian for "uint16"/"uint32"/"uint64"/"int16"/"int32"/"int64"/"single"/"double"
- Long Frames — true means we will transmit long frame data. It has the same data type for long frame data.

- The 1st frame Number — The 1st byte in data field for any long frame is frame number. For the 1st frame, the frame number can begin in number 0 or number 1.
- The last frame DLC keeps 8 — All frames' DLC is 8 except the last frame. The DLC of the last frame can be 8 or can be less than 8. If we keep all frames' DLC =8, the remaining data can be stuffed with 0.

Ports

Input

- Data — "double" data type's vector. This data type will be changed to parameter "Data type" in the final transmitted data field. This Data vector will be transmitted out in CAN Bus data frame.

Outpot

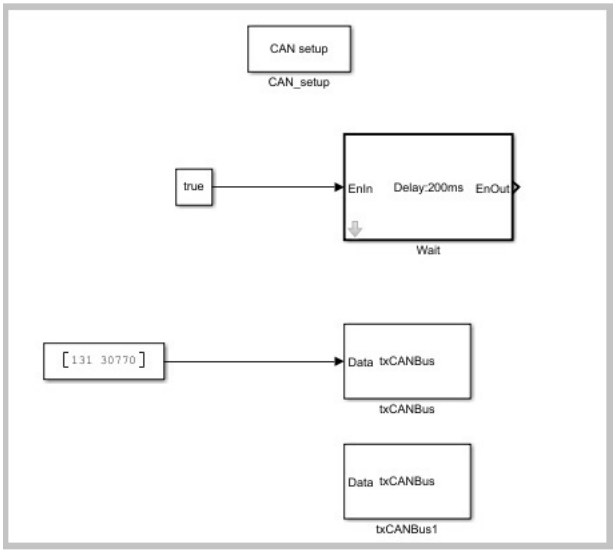
None

Examples

Example1:

Every 200ms, Action1: we will transmit single Data frame with 11 bits's standard CAN ID=68, and DLC=3, the first Data is byte "0x83", the second Data (uint16) is "0x7832" (Little-Endian).
Action2: we will transmit RTR frame with 29 bits's extended CAN ID=86

Please see screenshot of model below:



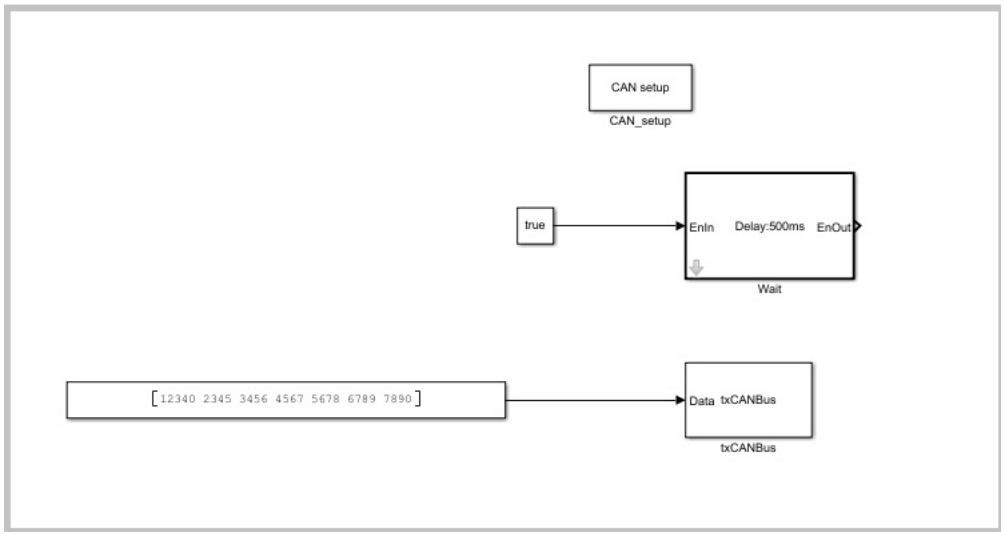
Please open "your CAN Controller library folder"/examples/example1_transmitSingle.slx (You must

change USB serial Port number in CAN_setup block according to your physical USB port number)

Example2:

Every 500ms, we will transmit long frames with 7 data items, each item will change to "int32". First frame number is 0, and the last frame DLC=8. The 7 data items are "12340, 2345, 3456, 4567, 5678, 6789, 7890". CAN Bus ID is standard "678".

Please see screenshot of model below:



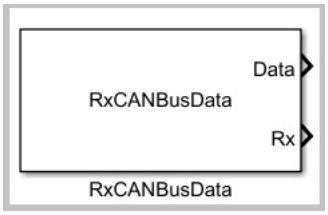
Please open "your CAN Controller library folder"/examples/example2_transmitLong.slx (You must change USB serial Port number in CAN_setup block according to your physical USB port number)

4.3 rxCANBusData

rxCANBusData

receive CAN Bus Data frame
Since R2019b

Library: CAN Bus Controller (Dafulai Electronics) /rxCANBusData



Description

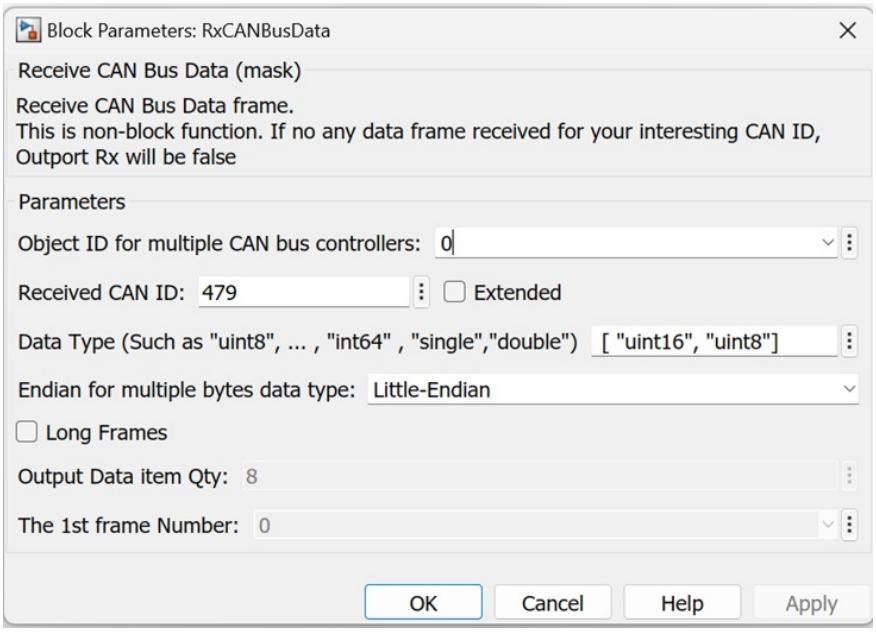
This block receives single CAN Data frame, and long Data frames. This is non-block function. It will return immediately no matter whether it receives your interesting CAN ID.

For single Data frame, DLC is decided by Data Input vector.
For long Data frames, frame Qty is decided by Data Input vector.

If "long Frames" is received, you must make sure all long frames are ready in receiver buffers.
How to make sure? you must use "catchCANID" block to catch "the last frame of long data frames".
And one solution is Simulink model's truly sampling time (Not simulated time) must be bigger than entire long frames' received time.
Please use "wait" block to implement truly sampling time. The other solution is that disable auto-refresh in "CAN Bus setup" block. When caught "the last frame of long data frames" by "catchCANID" block, call the "wait" block to delay "Long frames's broadcast cycle time" (Real time, not simulated time).

Parameters

Please double click this block to open parameters dialog below:



Let us explain parameters.

- Object ID for multiple CAN Bus controllers — In one PC, we may use multiple CAN bus controllers, this is for identifying each one we use. It must match the same parameter in CAN_setup block.
- Received CAN ID — received CAN bus data frame's CAN ID. It is scalar.
- Extended — true means "Received CAN ID" is 29 bits of extended. Otherwise, it is 11 bits of

standard.

- Remote Request Frame — true means "Transmit RTR frame". Otherwise, it means "Transmit Data frame".
- Data Type — CAN bus data field's Data type. For single data frame, It is vector. The element of this vector is string. The valid string is "uint8", "uint16", "uint32", "uint64", "int8", "int16", "int32", "int64", "single", "double". It denotes "each data element's Data type in CAN bus frame's data field" although all elements' data type in output port "Data" vector is "double" (transfer to "double" for user easily to read). For long data frames, it is scalar string because all data type is the same for long frames.
- Endian for multiple bytes data type — Big-Endian or Little-Endian for "uint16"/"uint32"/"uint64"/"int16"/"int32"/"int64"/"single"/"double"
- Long Frames — true means we will receive long frame data. It has the same data type for long frame data.
- Output Data item Qty — Long frames' all truly data items Qty in your setting Data type.
- The 1st frame Number — The 1st byte in data field for any long frame is frame number. For the 1st frame, the frame number can begin in number 0 or number 1.

Ports

Input

None

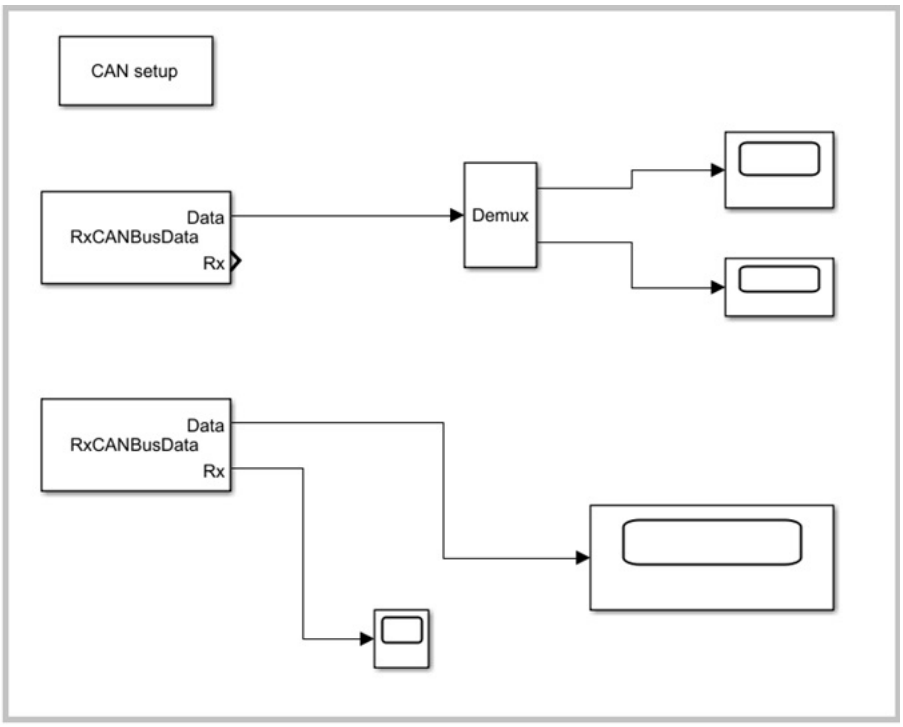
Outport

- Data — "double" data type's vector. It is all received Data which are from received CAN bus raw data with your setting Data Type in input parameter. If we didn't receive any data in this sample time, the output port Data will keep previous Data Value.
- Rx — "logical" scalar. True means we received data in this sample time.

Examples

Example1:

We are interesting in CAN ID=888 standard data frame, and CAN ID=999 extended data frame.
It has 2 data, one is "uint16" (Little-Endian) and the other is "uint8" in CAN ID=888 standard data frame. So DLC will be 3 automatically.
It has "int16" (Big-Endian) in CAN ID=999 extended data frame. So DLC will be 2 automatically.
Please see screenshot of model below:

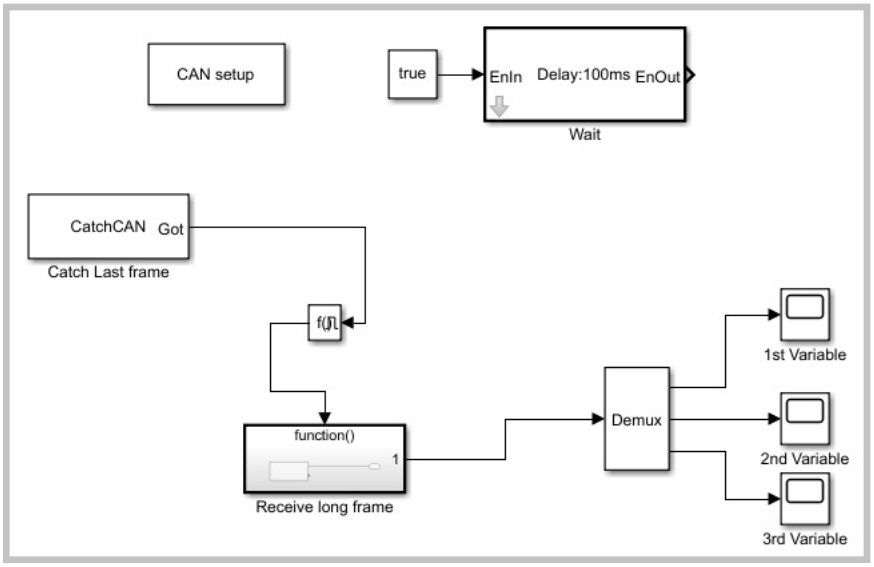


Please open "your CAN Controller library folder"/examples/example1_receiveData.slx (You must change USB serial Port number in CAN_setup block according to your physical USB port number)

Example2:

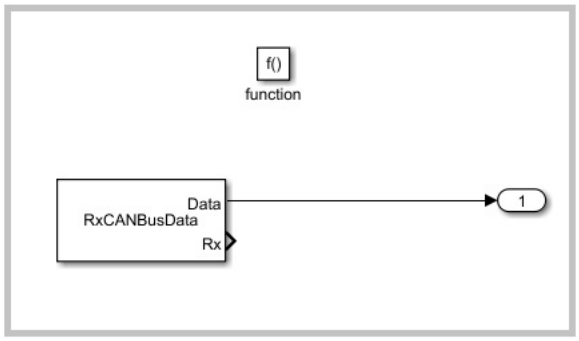
Every 300ms, long frames (CAN ID =555 standard data frame) will be coming. It has 3 data items of int32 type with Little-Endian. The first frame number is 0,

Please see screenshot of model below:



It has "Wait" 100ms block (we make assumption: long frames will be done within 100ms). This "wait" block will lead to actual sampling time=100ms besides simulated sampling time=0.1 sec.
Please see "catchCANID" block help in library. "catchCANID" block will catch the last frame of CAN ID=555 standard long frames, and create one sampling time pulse.
This pulse will create a function call signal by function call generator block. This signal triggers "Function call sub-system" ("Receiver long frame" block) to be called once.

Please see inside of "Function call sub-system" below:



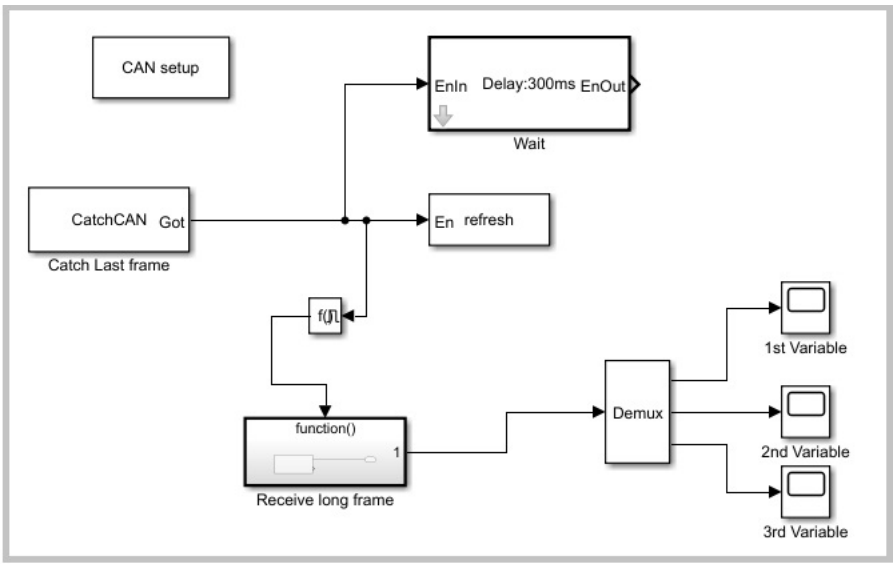
"RxCANBusData" block will get long frames' data output. And at last , we use Demux block to divide vector to 3 variables which are displayed in 3 Scopes.

Please open "your CAN Controller library folder"/examples/example2_receiveData.slx (You must change USB serial Port number in CAN_setup block according to your physical USB port number)

Example3:

The same as Example2. However, we disable "Auto-refresh" in "CAN setup" block. We use "refresh" block manually when the last frame of long data frames coming. In example2, it has risk of missing some long frames if sampling time is exact between the middle of all long frames. This example 3 can avoid this situation.

Please see screenshot of model below:



.
In CAN setup block, we disabled "Auto-refresh" function. And when we caught the last frame of "555" data frames, CatchCAN block will output one pulse. This pulse will give to 3 blocks.
One block is 300ms wait which is the same as "long frames" broadcast period. And the 2nd block will create refresh. So all receiver frames will be updated. The 3rd block will create one "function call signal".This signal triggers "Function call sub-system" ("Receiver long frame" block) to be called once. The inside of "Function call sub-system" is the same as one in example2. "RxCANBusData" block will get long frames' data output. And at last , we use Demux block to divide vector to 3 variables which are displayed in 3 Scopes.

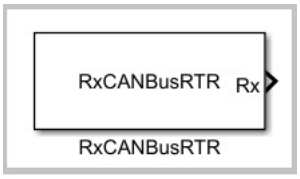
Please open "your CAN Controller library folder"/examples/example3_receiveData.slx (You must change USB serial Port number in CAN_setup block according to your physical USB port number)

4.4 rxCANBusRTR

rxCANBusRTR

receive CAN Bus Data frame
Since R2019b

Library: CAN Bus Controller (Dafulai Electronics) /rxCANBusRTR

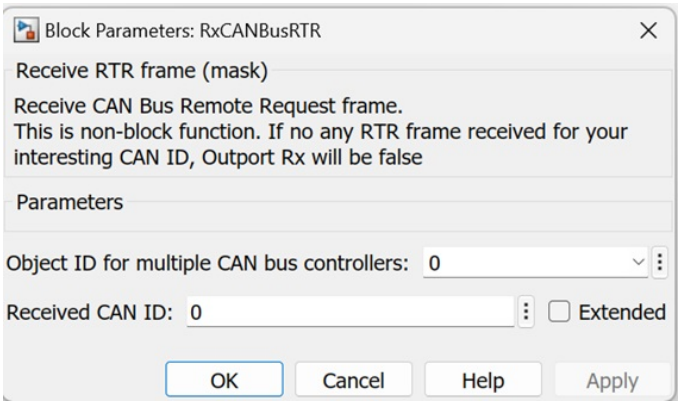


Description

This block receives one CAN RTR frame.. This is non-block function. It will return immediately no matter whether it receives your interesting CAN ID. The output port "Rx" (logical value) will tell you whether received RTR frame.

Parameters

Please double click this block to open parameters dialog below:



Let us explain parameters.

- Object ID for multiple CAN Bus controllers — In one PC, we may use multiple CAN bus controllers, this is for identifying each one we use. It must match the same parameter in CAN_setup block.
- Received CAN ID — Received CAN bus frame's CAN ID. It is scalar.
- Extended — true means "Received CAN ID" is 29 bits of extended. Otherwise, it is 11 bits of standard.

Ports

Input

None

Outport

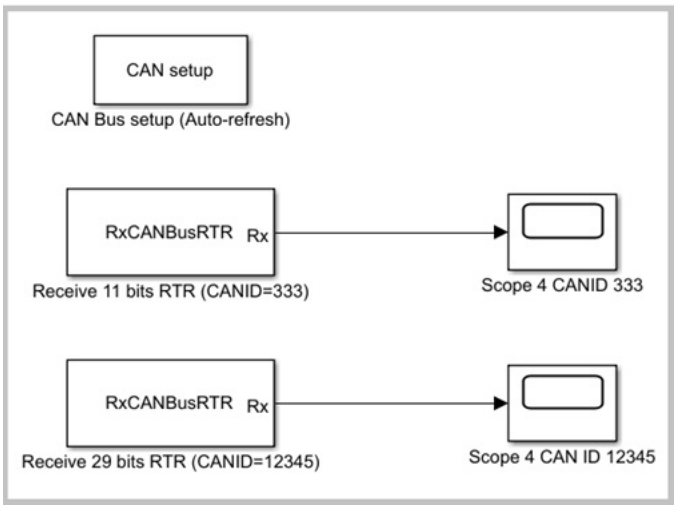
Rx — "logical" scalar. True means we received RTR frame in this sample time.

Examples

Example1:

We are interesting in CAN ID=333 standard RTR frame, and CAN ID=12345 extended RTR frame.

Please see screenshot of model below:



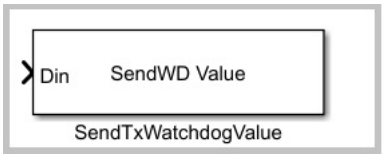
Please open "your CAN Controller library folder"/examples/example1_receiveRTR.slx (You must change USB serial Port number in CAN_setup block according to your physical USB port number)

4.5 SendTxWatchdogValue

SendTxWatchdogValue

send TxWatchdog Counter Value
Since R2019b

Library: CAN Bus Controller (Dafulai Electronics) /SendTxWatchdogValue

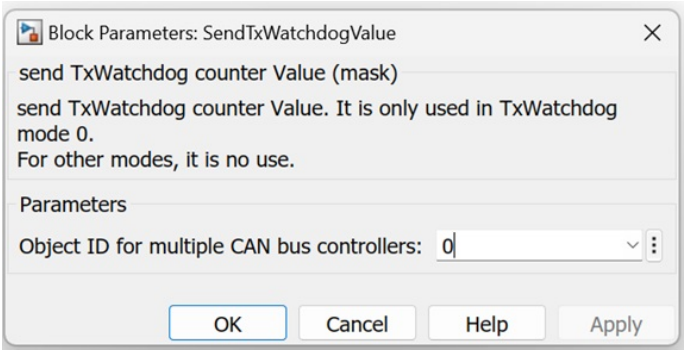


Description

This block will set Hardware TxWatchdog counter value for Txwatchdog Mode 0. If TxWatchdog is not enabled, system just ignore this block.

Parameters

Please double click this block to open parameters dialog below:



Let us explain parameters.

- Object ID for multiple CAN Bus controllers — In one PC, we may use multiple CAN bus controllers, this is for identifying each one we use. It must match the same parameter in CAN_setup block.

Ports

Input

Din — "number" scalar. Value Range is from 2 bytes of integer. If Counter is one byte, it will only use LSB automatically,

Outport

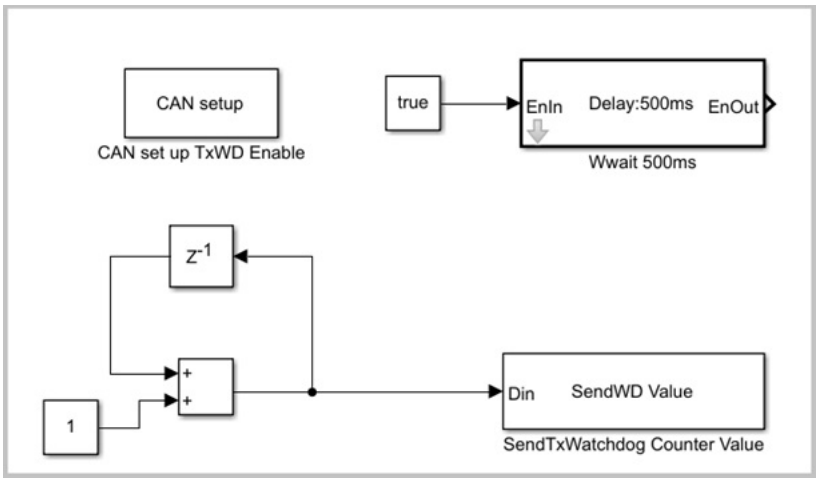
None

Examples

Example1:

We use txWatchdog Mode 0 and TxWatchdog periods=500ms. TxWatchdog frame DLC=8. TxWatchdog coounter has 2 bytes starting from the 1st byte of frame's data field. Our PC will be in charge of setting TxWatchdog Counter value every 500ms. And the Counter value will increase 1 every 500ms.

Please see screenshot of model below:



It has "Wait" 500ms block. This "wait" block will lead to actual sampling time=500ms besides simulated sampling time=0.5 sec.

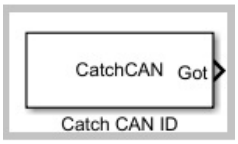
Please open "your CAN Controller library folder"/examples/example1_sendTxWatchdog.slx (You must change USB serial Port number in CAN_setup block according to your physical USB port number)

4.6 catchCANID

catchCANID

catch special CAN Bus frame
Since R2019b

Library: CAN Bus Controller (Dafulai Electronics) /catchCANID



Description

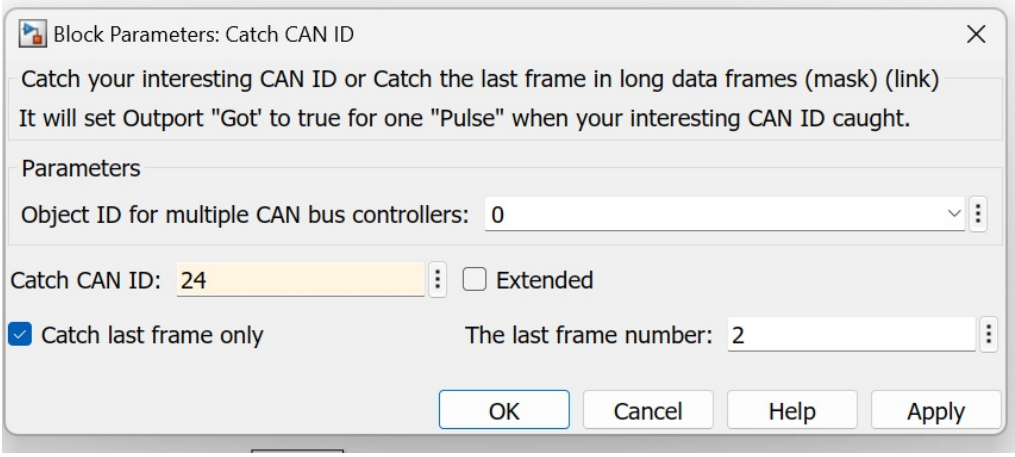
This block catches your interesting CAN Bus frame. The interesting CAN Bus frame is divided into 2 different types. Type 1 is that we only care CAN ID. Type 2 is that we care not only CAN ID but also first byte of data field (the frame number of long data frames).
This is non-block function. It will return immediately no matter whether it catch your interesting CAN bus frame. The output port "Got" (logical value) will tell you whether catch your interesting CAN Bus frame.

***Notes**: long data frames means that CAN ID is the same, and the first byte of data field is frame number (starting from 0 or 1). So data field of each frame only has 7 bytes truly data, Total maximum data Qty= 7 x 256 (first frame No=0)=1792 bytes Or Total*

maximum data Qty= 7 x 255 (first frame No=1)=1785 bytes

Parameters

Please double click this block to open parameters dialog below:



Let us explain parameters.

- Object ID for multiple CAN Bus controllers — In one PC, we may use multiple CAN bus controllers, this is for identifying each one we use. It must match the same parameter in CAN_setup block.
- Catch CAN ID — It is CAN bus frame's CAN ID you want to catch. It is scalar.
- Extended — true means "Catch CAN ID" is 29 bits of extended. Otherwise, it is 11 bits of standard.
- Catch last frame only — true means we catch the CAN Bus data frame which CAN ID is specified in above parameters, and also which is the last frame of long data frames. Otherwise, we only care CAN ID/Extended.
- The last frame number — The frame number of the last frame in long data frame.

Ports

Input

None

Outport

Got — "logical" scalar. True means we caught our interesting CAN Bus frame.

Examples

Example1:

We are interesting in CAN ID=44 standard frame. We'd like to catch the last frame with the 1st byte of data field being equal to 2.

Please see screenshot of model below:

If you send CAN bus data frame with CAN ID =44 (11 bits CAN ID) and 8 Data bytes=0x2, 0x0, 0x0,0x0, 0x0, 0x0, 0x0, 0x0 by external CAN Bus device, you will see one pulse in Simulink Scope from above model.

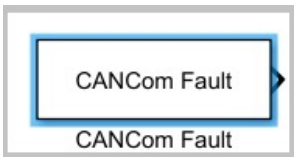
Please open "your CAN Controller library folder"/examples/example1_catchCANID.slx (You must change USB serial Port number in CAN_setup block according to your physical USB port number)

4.7 CAN_ComFault

CAN_ComFault

get CAN bus communication fault state according to its RxWatchdog
Since R2019b

Library: CAN Bus Controller (Dafulai Electronics) /CAN_ComFault



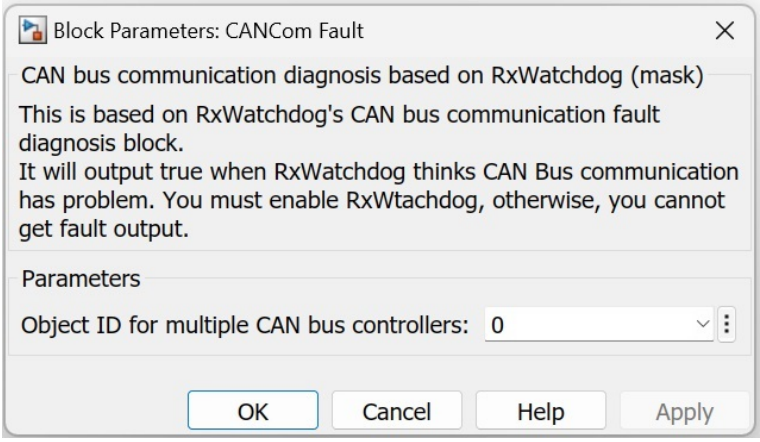
Description

This block will indicate whether CAN Bus communication fault occurs according to its RxWatchdog. You must enable RxWatchdog in CAN_setup block in order to get correct communication state. Otherwise, we will get "always communication normal" (Output port= false)

> ***Notes**: This block has nothing to do with refresh. It will output fault state even though you didn't run any refresh (Auto-refresh or manually refresh block)*

Parameters

Please double click this block to open parameters dialog below:



Let us explain parameters.

- Object ID for multiple CAN Bus controllers — In one PC, we may use multiple CAN bus controllers, this is for identifying each one we use. It must match the same parameter in CAN_setup block.

Ports

Input

None

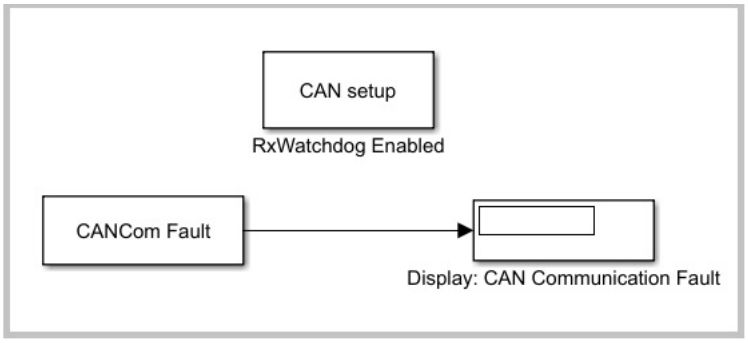
Outport

Fault — "logical" scalar. True means CAN bus communication fault.

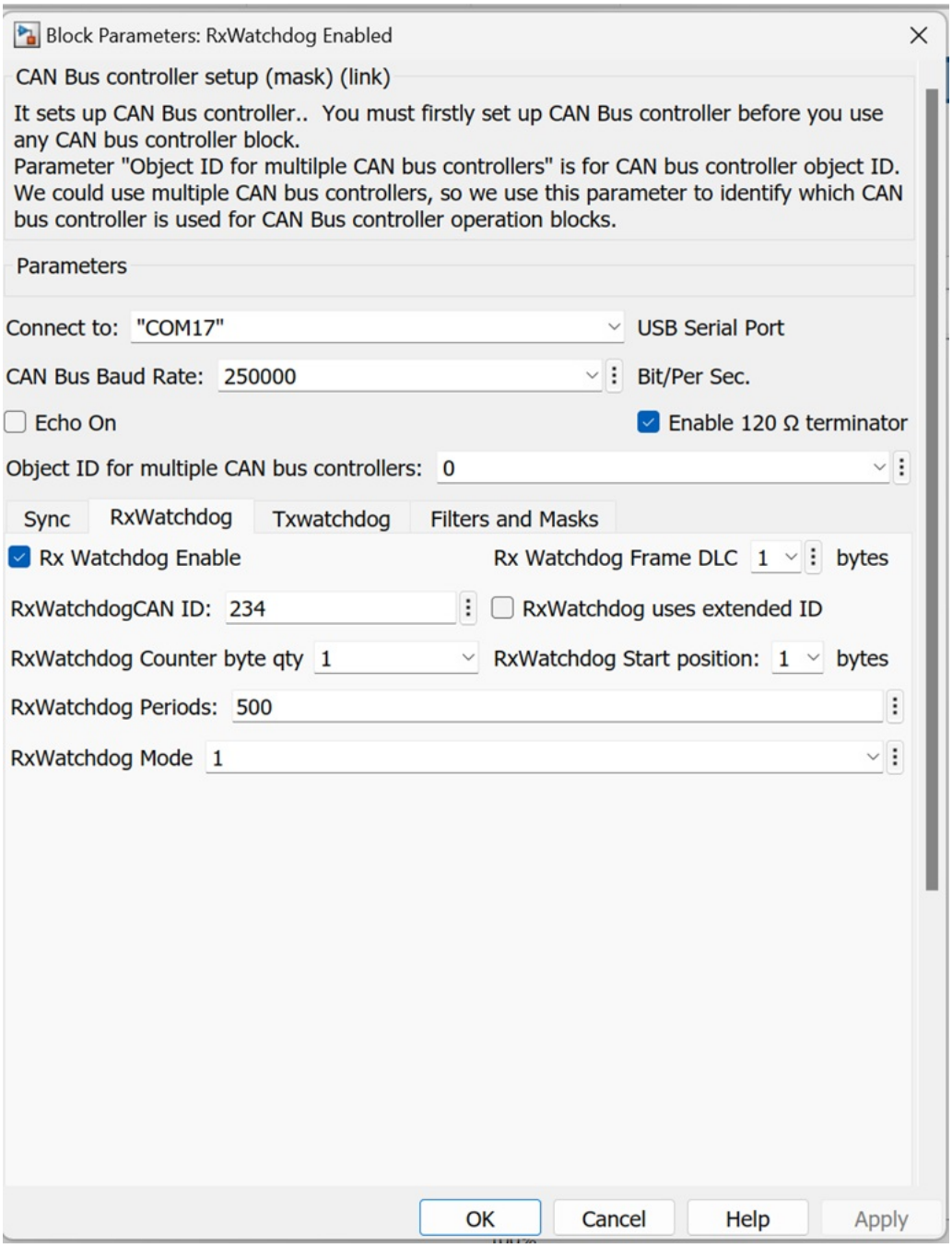
Examples

Example1:

The following model will detect CAN Bus communication fault:



We will Enable CAN Bus RxWatchdog in CAN_setup block. Please see parameters settings in "CAN_setup" block below:



So if we don't send out data frame with CAN ID=234 from external CAN Bus device, we will get value 1 from Display block (Communication fault).
However, if we send out data frame with CAN ID=234 and Data1=0 (first byte in data field: RxWatchdog counter) and DLC=1 every 330ms from external CAN Bus device, we will get value 0 from Display block (Communication normal).

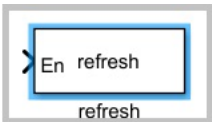
Please open "your CAN Controller library folder"/examples/example1_CAN_ComFault.slx (You must change USB serial Port number in CAN_setup block according to your physical USB port number)

4.8 refresh

refresh

refresh CAN bus receiver buffers
Since R2019b

Library: CAN Bus Controller (Dafulai Electronics) /refresh

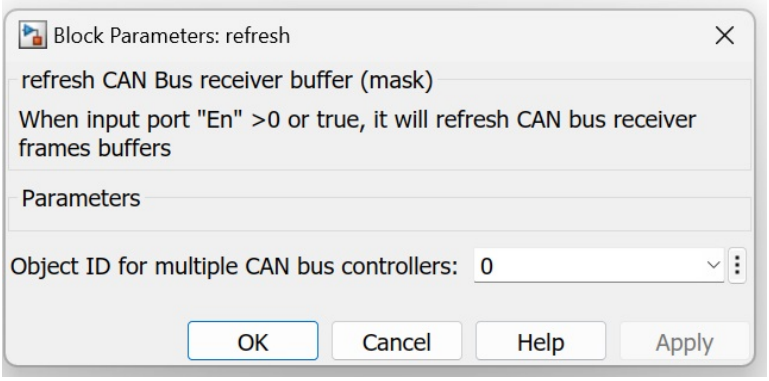


Description

This block will refresh CAN Bus receiver buffers. After you executing this block, the previous CAN bus receiver information will disappear.
Usually, we don't use this block because our CAN_setup block can set up "Auto-refresh" enabled.
We use this block when "Auto-refresh" disabled.

Parameters

Please double click this block to open parameters dialog below:



Let us explain parameters.

- Object ID for multiple CAN Bus controllers — In one PC, we may use multiple CAN bus controllers, this is for identifying each one we use. It must match the same parameter in CAN_setup block.

Ports

Input

En — "number" scalar or "logical" scalar. true or >0 will make once refresh in this sampling time.

Outport

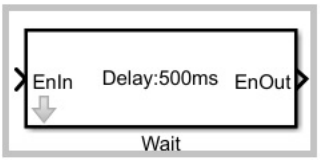
None

4.9 wait

wait

wait some time to pass.
Since R2019b

Library: CAN Bus Controller (Dafulai Electronics) /wait



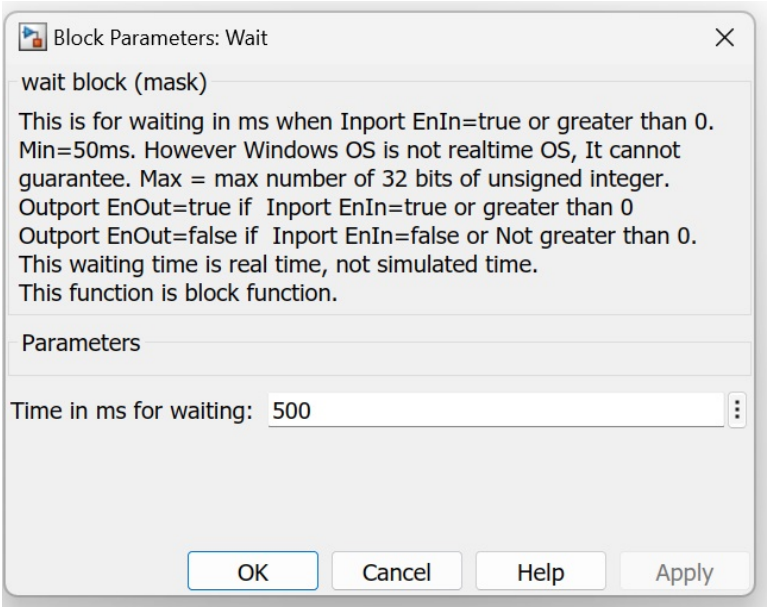
Description

This block will wait some milliseconds. This is block-function, it is different from simulated-sampling time, it is truly time. After the truly time passed, this block completes and it can run other remaining blocks in the entire model.

> ***Notes***: *Windows/Linux/MacOS is not real time OS. So this block cannot guarantee real time.*

Parameters

Please double click this block to open parameters dialog below:



Let us explain parameters.

- Time in ms for waiting — Delay time in milliseconds.

Ports

Input

EnIn — "number" scalar or "logical" scalar. true or >0 will delay specified time. false will be no any delay.

Outport

EnOut — "logical" scalar. It is EnIn value. If EnIn is "number" type, EnOut= true when EnIn>0.

5 Notice

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