Project: CANBUS – Priority Management Using Canoe

TEAM : Daboussi Mariem	Chaari Mahdi
Bargaoui Nourchène	Nasri Moahmed Youssef
Jelidi Amel	Jebari Houssem
Karoui Nouha	Ben salah mehdi ismail

Module : RLI	**Class :** IIA4
Professor : Hamdi Mustapha	**Academic year :** 2020/2021

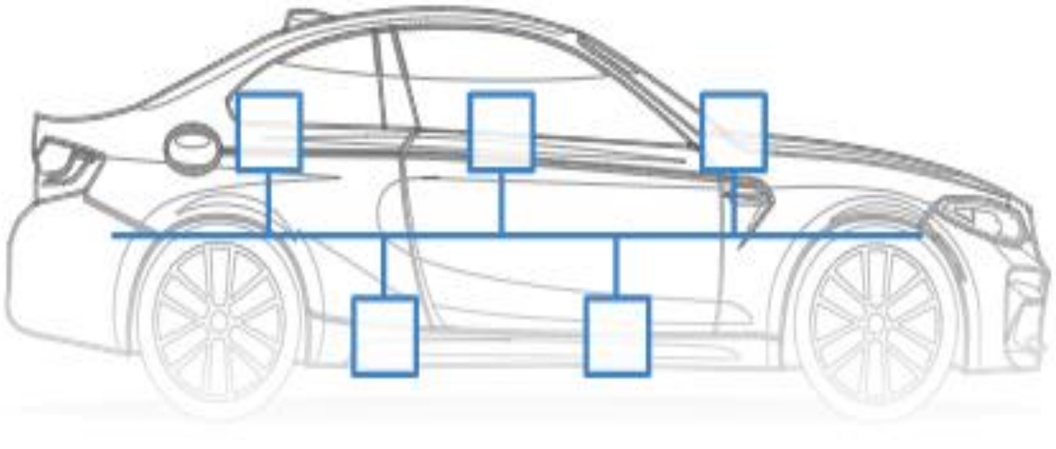


Table of Content :

1.Chapter 1 : CAN_BUS :

1.1-Introduction to Can-Bus:

Your car is like a human body:

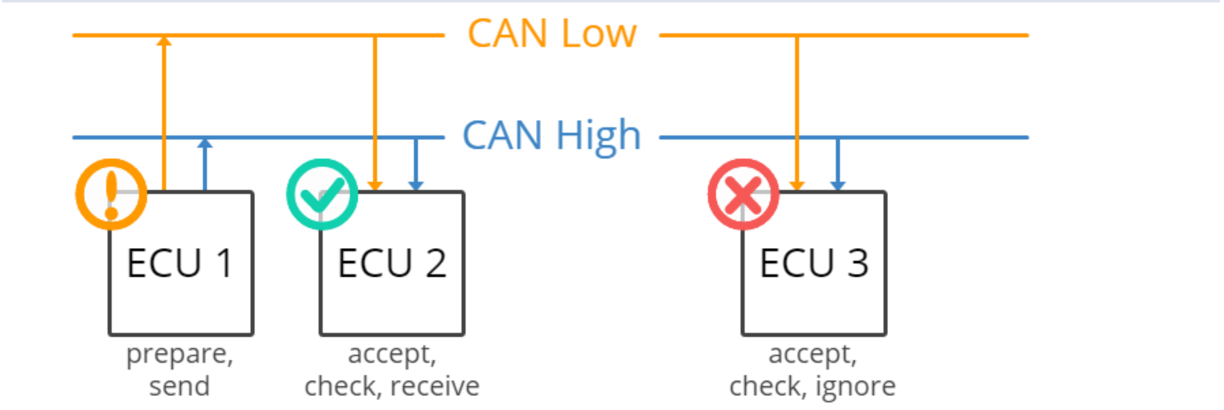
The Controller Area Network (CAN bus) is the **nervous system**, enabling communication.

In turn, 'nodes' or 'electronic control units' (ECUs) are like parts of the body, interconnected via the CAN bus. Information sensed by one part can be shared with another.

So what is an ECU?

In an automotive CAN bus system, ECUs can e.g. be the engine control unit, airbags, audio system etc. A modern car may have **up to 70 ECUs** - and each of them may have information that needs to be shared with other parts of the network.

This is where the CAN standard comes in handy:

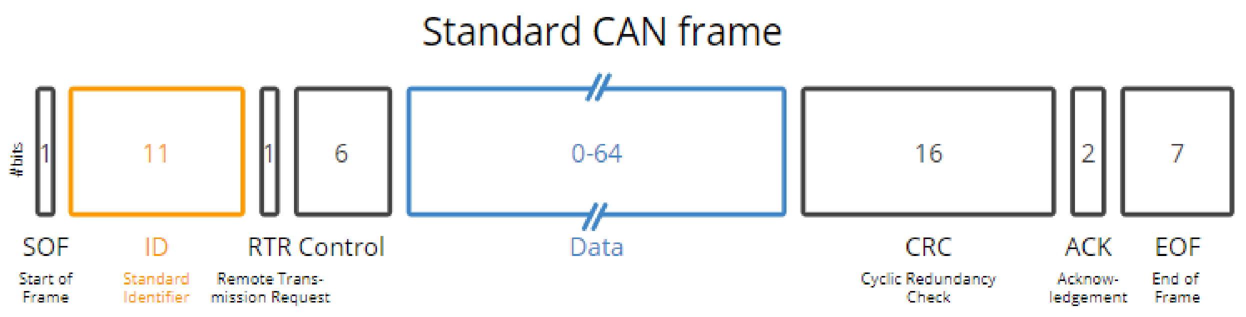


The CAN bus system enables each ECU to communicate with all other ECUs - without complex dedicated wiring.

Specifically, an ECU can prepare and broadcast information (e.g. sensor data) via the CAN bus (consisting of two wires, CAN low and CAN high). The broadcasted data is accepted by all other ECUs on the CAN network - and each ECU can then check the data and decide whether to receive or ignore it.

1.2-The CAN DataFrame :

Communication over the CAN bus is done via CAN frames as shown below:



The 8 CAN bus protocol message fields

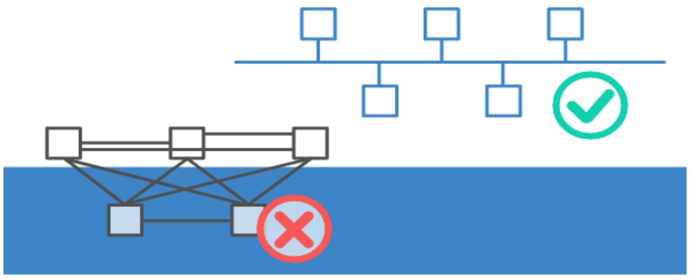
It's important to note that there is no explicit address in the CAN messages. Each CAN controller will pick up all traffic on the bus, and using a combination of hardware filters and software, determine if the message is "interesting" or not.

In fact, there is no notion of message addresses in CAN. Instead, the contents of the messages is identified by an identifier. CAN messages are said to be "contents-addressed".

A conventional message address would be used like "Here's a message for node X". A contents-addressed message is like "Here's a message containing data labeled X". The difference between these two concepts is small but significant.

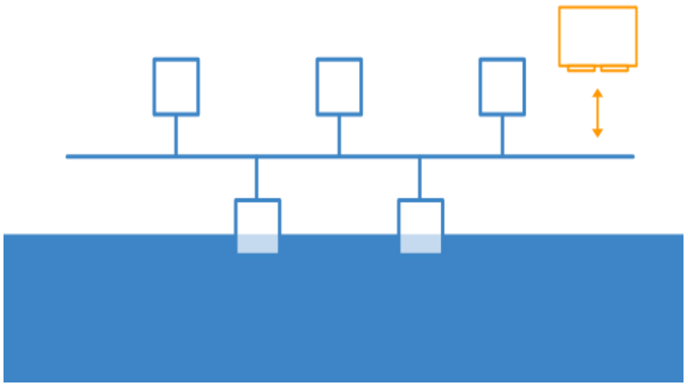
The contents of the Arbitration Field is, per the Standard, used to determine the message's priority on the bus.

1.3-Benefits of Can-Bus :



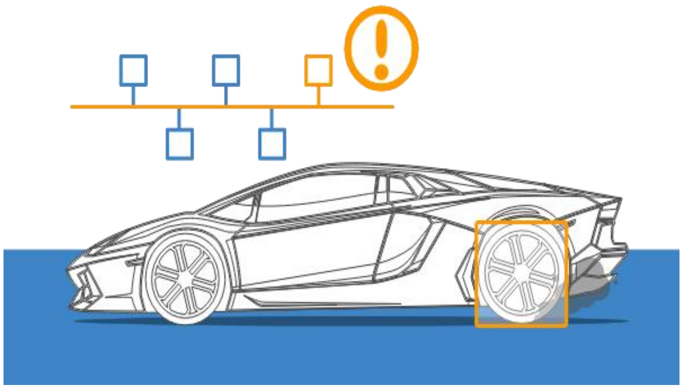
Simple & low cost

ECUs communicate via a single CAN system instead of via direct complex analogue signal lines - reducing errors, weight, wiring and costs.



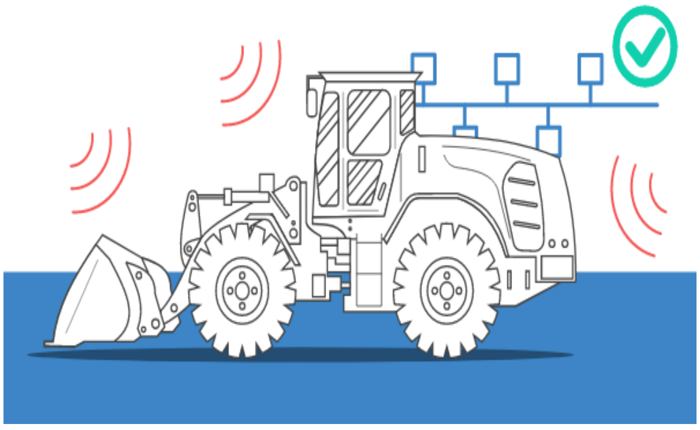
Fully centralized

The CAN bus provides 'one point-of-entry' to communicate with all network ECUs - enabling central diagnostics, data logging and configuration.



Extremely robust

The system is robust towards electric disturbances and electromagnetic interference - ideal for safety critical applications (e.g. vehicles)



Efficient

CAN frames are prioritized by ID so that top priority data gets immediate bus access, without causing interruption of other frames

2-Chapter 2: Application:

Part I : About our project

Our project focuses on displaying how the CANOE VECTOR is a great tool to for creating a CAN bus as a standard for connecting the comfort equipments of a vehicle (in our case window regulators) and how changing the ID of messages determines their priority .

Part II : The SetUp

Before opening CANoe, create a new directory called “testTFS” to contain all the files pertaining to this tutorial. This folder will have additional folders for better understanding and organized access of the files created by CANoe. Create the following sub-folders:

1) CANoe Config
2) DBC
3) ECU Simulation
4) Panels
5) Test Files
6) Test Reports
7) TestUnit Build Output
8) vTESTstudio Project

• CAN database

A network database is not required, but it is always helpful to have one. In most test applications, a network database will save a lot of time and effort in implementing the test cases. If one is not available, creating a database is a good first step.

• Create new file

In CANoe, click the “Tools” ribbon and select CANdb++ Editor to launch the integrated CAN database editor. In the CANdb++ Editor, go to File | Create Database. A prompt appears to select a template. Select the template named "EmptyTemplate.dbc". Name the empty DBC file "testTFS.dbc" and save it into the “DBC” sub-folder

• Create ECU nodes

Select “Network Nodes” from the tree view on the left pane of the Overall View window, right-click on it, and select New. Enter "Doors" in the Name textbox and then click OK

⇨ Create three network nodes, “Engine”, “Console” and “Doors”

• Create messages

Create five messages for our simple test application.

1) Select "Messages" in the tree view on the left pane of the Overview window, right-click on it, and select New.

2) Setup the messages as follows:

- Name = EngineStatus

• Set the DLC (Data LengthCode) to “2”, for 2 bytes of data.

• Select the Transmitters tab and click [Add]. Select “Engine” and click [OK].

-Name = LockingRq

• Set the DLC to “1”.

• Set “Console” to be the transmitter of the message.

-Name = LockingState

• Set the DLC to “1”.

• Set “Doors” to be the transmitter of the message. o Name = WindowRq

• Set the DLC to “1”.

• Set “Console” to be the transmitter of the message.

-Name = WindowState

• Set the DLC to “1”

• Set “Doors” to be the transmitter of the message.

• Create signals

Create six signals to represent data within the messages we have just created as follows:

1) Select “Signals” in the tree view on the left pane of the Overview window, right-click on it and select New.

2) Setup the signal as follows:

Name = IgnitionStatus

Name = LockRequest

Name = LockState

Name = Velocity

Name = WindowPosition

Name = WindowRequest

• Create value tables

Value tables can be made for each signal to represent symbolically, the different value states. We will create a value table for five of our signals.

1) Select View | Value Tables.

2) Right-click anywhere on the empty page and select New to create a value table.

3) Setup the value tables as follows:

o Name = VtSig_IgnitStatus

• In the Value Descriptions tab, click the [Add] button twice.

        - Replace the description for “0x0” to “Engine Off”
        - Replace the description for “0x1” to “Engine On”
    - Name = VtSig_LockRequest
        - In the Value Descriptions tab, click the [Add] button twice.
        - Replace the description for “0x0” to “Request to Lock”
        - Replace the description for “0x1” to “Request to Unlock”
    - Name = VtSig_LockState
        - In the Value Descriptions tab, click the [Add] button twice.
        - Replace the description for “0x0” to “Locked”
        - Replace the description for “0x1” to “Unlocked”
    - Name = VtSig_WindowPosition
        - In the Value Descriptions tab, click the [Add] button twice.
        - Replace the description for “0x0” to “Rolled Up”
        - Replace the description for “0xF” to “Rolled Down”
    - Name = VtSig_WindowRequest
        - In the Value Descriptions tab, click the [Add] button thrice.
        - Replace the description for “0x0” to “Nothing”
        - Replace the description for “0x1” to “Roll Window Up”
        - Replace the description for “0x2” to “Roll Window Down”

4) Go back to the Overview window.

5) Double-click on “IgnitionStatus" under Signals. In the window that appears, select “VtSig_IgnitStatus” in the drop-down box for Value Table. Click [OK].

6) Double-click on “LockRequest" under Signals. In the window that appears, select “VtSig_LockRequest” in the drop-down box for Value Table. Click [OK].

7) Double-click on “LockState" under Signals. In the window that appears, select “VtSig_LockState” in the drop-down box for Value Table. Click [OK].

8) Double-click on “WindowPosition" under Signals. In the window that appears, select “VtSig_WindowPosition” in the drop-down box for Value Table. Click [OK].

9) Double-click on “WindowRequest" under Signals. In the window that appears, select “VtSig_WindowRequest” in the drop-down box for Value Table. Click [OK]

• Associate the database

1) In CANoe select the “Simulation” ribbon and click on Simulation Setup to make sure the Simulation Setup is displayed.

2) Expand the tree list on the right pane and select Databases. Right-click on it and select Add.

3) Navigate to the DBC sub-folder and select the “testTFS.dbc” database file.

• Setting up CANoe

The CANoe environment needs to be setup so that all the components can interact with each other. The simulated ECU and the remaining bus simulation must be created using CAPL. The system variables are like global variables. In this tutorial, we will use the System Variables to control the Engine and Console ECU simulation from other components like the Panel and Test Units.

• Add nodes to the network

Go to CANoe and open the Simulation Setup. In the left pane of the window, right-click on the blue/red parallel lines and select Insert Network Node. Name it “Doors”. Add 2 more nodes “Engine” and “Console”

• Create simulated doors ECU

Next, we will use the CAPL programming language of CANoe to define the behavior of our simulated Doors ECU. A CAPL program is usually developed using the CAPL Browser. The CAPL Browser can be accessed from the “Tools” ribbon

1) In the Simulation Setup of CANoe click on the pencil icon located in the lower-left corner of the “Doors” network node.

2) An Open dialog will appear asking for the CAPL program to be given a name. Type in “doors.can” and navigate to the “ECU Simulation” subfolder. Then, click on Open

```
variables
{
  byte WindowState = 0;
  byte IgnitionState;

  message LockingState LockingStat;
  message WindowState WindowStat;
}
```

3) In the tree view of the left pane of CAPL browser, right-click on “CAN” and select the option “New Event Handler” | “on message”. An empty procedure for a CAN message appears in code area.

4) Right-click on the highlighted NewMessage and select the option “Insert Database Symbol” | “Message or Frame”.

5) Select the CAN frame “EngineStatus” from the tree view under Frames.

6) Enter the following code:

```
on message EngineStatus
{
  // Storing Ignition State
  if(this.IgnitionStatus) IgnitionState = 1; // Ignition is On
  else IgnitionState = 0; // Ignition if Off

  // Locking based on velocity
  if(IgnitionState == 1 && this.Velocity > 15 &&
LockingStat.LockState == 1)
  {
    LockingStat.LockState = 0;
    output(LockingStat);
  }
}
```

7)) Select the CAN frame “WindowRq”

```
on message WindowRq
{
  if(IgnitionState)
  {
    if(this.WindowRequest == 1) //Roll Up window
      if(WindowState > 0) WindowState--;
    if(this.WindowRequest == 2) //Roll Down window
      if(WindowState < 15) WindowState++;
  }
  WindowStat.WindowPosition = WindowState;
  output(WindowStat);
}
```

• Create system variables

1) In CANoe, select the Environment ribbon and click on the “System Variables”. The System Variables Configuration window will open.

2) In the empty left pane, right-click and select New.

3) Enter the following details for creating a System Variable

4) We will now create 4 more System Variables by right-clicking on the “testNS” row and selecting New.

Variable	Name	Initial Value	Minimum	Maximum
1	“IgnitionStart”	0	0	1
2	“LockRq”	0	0	1
3	“Velocity”	0	0	200
4	“WindowRequest”	0	0	2
5	“WindowState”	0	0	15

• Add remaing bus simulation

Next, we will again use the CAPL programming language to define the behavior of the remaining bus. The following program is written based on the expectations of the Doors as specified in the Appendix. 1) In the Simulation Setup of CANoe click on the pencil icon located in the lower-left corner of the “Engine” network node. 2) An Open dialog will appear asking for the CAPL program to be given a name. Type in “engine.can” and navigate to the “ECU Simulation” subfolder.

```
variables
{
  //The following three messages are defined for transmission
  message EngineStatus EngineStat;
  message LockingRq LkCtrlRq;
  message WindowRq WindowCtrl;
  //The following timer is for simulating the cyclic message
transmission
  msTimer msTimer_EngineStatus;
}
```

In the tree view of the left pane, right-click on System and select the option “New Event Handler | on start”. Enter the following code:

```
on start
{
  setTimerCyclic(msTimer_EngineStatus,100);
}
```

```
on timer msTimer_EngineStatus
{
  EngineStat.Velocity = @sysvar::testNS::Velocity;
  EngineStat.IgnitionStatus = @sysvar::testNS::IgnitionStart;
  output(EngineStat);
}
```

In the Simulation Setup of CANoe click on the pencil icon located in the lower-left corner of the “Console” network node. An Open dialog will appear asking for the CAPL program to be given a name. Type in “console.can” and navigate to the “ECU Simulation” subfolder

In the tree view of the left pane, right-click on Value Objects and select the option “New Event Handler | on sysvar_update”. In the Symbols window, click on the “Variables” icon and navigate to LockRq variable. Now drag and drop the variable on the highlighted “NewSysVar”. Enter the following code :

```
on sysvar_update testNS::LockRq
{
  if(@sysvar::testNS::LockRq) LkCtrlRq.LockRequest = 1; // Request
Unlock Doors
  else LkCtrlRq.LockRequest = 0; // Request Lock Doors
  output(LkCtrlRq); // Outputs message onto the bus
}
```

In the tree view of the left pane, right-click on CAN and select “New Event Handler | on message”. Right-click on the highlighted NewMessage and select the option “Insert Database Symbol” | “Message or Frame”. Select the CAN frame “WindowState” from the tree view under Frames. Enter the following code:

```
on message WindowState
{
  @sysvar::testNS::WindowState = this.WindowPosition;
}
```

• User test panel

A Graphic panel can be created using the “Panel Designer”. This tool can be opened from the “Tools” ribbon from CANoe. It is recommended to open the tool from CANoe so that the associated database and system variables are referenced.

• Create new panel

In CANoe, go to the “Home” ribbon and click on “Panel | New Panel“. The Panel Designer tool will be opened

• Add controls

we will use 2 switches, 2 push buttons and 1 trackbar for controlling the variables. We will also use 2 indicators and 1 LCD control for display of outputs

1) On the right side of the Panel Designer, there should be a Toolbox window. If its not visible, go to the Home ribbon, and select Views Tool Box.

2) Scroll down the list of Vector Standard Controls and click on “Switch/Indicator” and drag it onto the platform in the center of Panel Designer. A box with a dashed border and a black color switch should then appear.

3) Open the “Properties” window using the Views option in the Home ribbon.

4) Under the Symbol section, set the symbol filter to Variable. Next, click on Symbol, and then click on the button to the right of the white box.

5) A window should appear with the list of Symbols, expand the System variable list, and select “IgnitionStart”. Afterwards, click [OK]. (Alternatively, the system variable can be assigned by dragging it from the Symbol Explorer on the left pane to the switch box.)

6) Under the Toolbox section, scroll down the list, drag and drop “Static Text” next to the switch on the gray platform. A box with the words “Description” should appear.

7) In the Properties box, replace the text box property “Text”, from “Description” to “Ignition Start”.
8) Repeat steps 2-5 for the following system variables:

o “LockRq”

o “WindowState” with state count of 17.

9) Repeat steps 2-5 again for the CAN signal “LockState”. For this, set the symbol fiter to Signal and select the signal “LockState” from the Symbol Selection window. Also, set the Display Only property to True

To add Push Buttons for window control:

1) From the Toolbox, drag and drop the “Button” control.

2) In the Properties window, change the Text from “Button” to “Window Up”.

3) Under the Symbol section, set the Symbol Filter to Variable. Now select the “WindowRequest” system variable in the Symbol option.

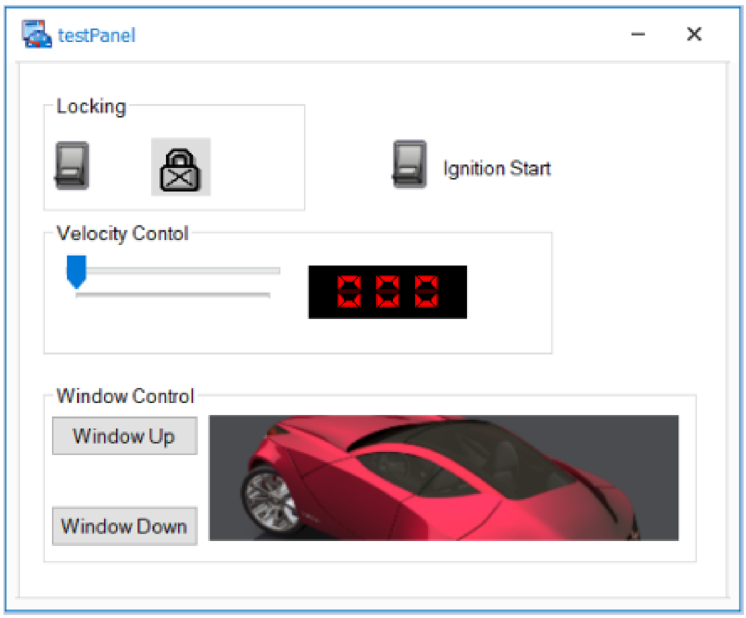
4) Under Switch Values option, select Roll_Up for Pressed and No_Request for Released.

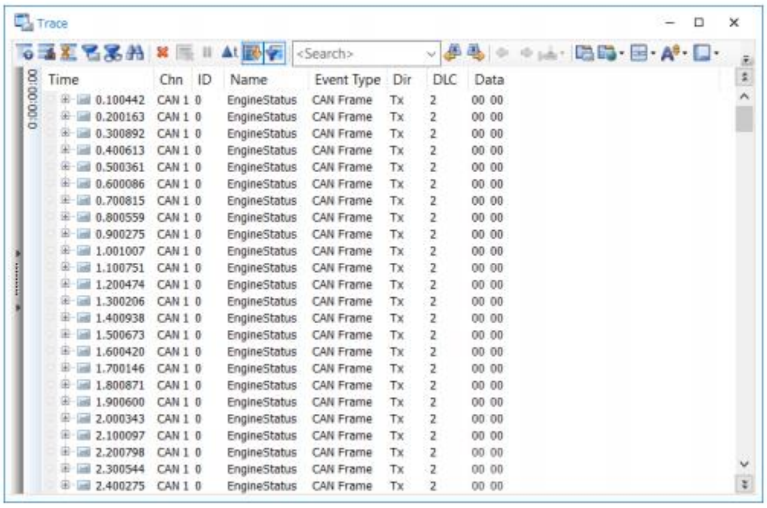
5) From the Toolbox, drag and drop another “Button” control.

6) In the Properties window, change the Text from “Button” to “Window Down”.

7) Under the Symbol section, set the symbol filter to Variable. Now select the “WindowRequest” system variable in the Symbol option.

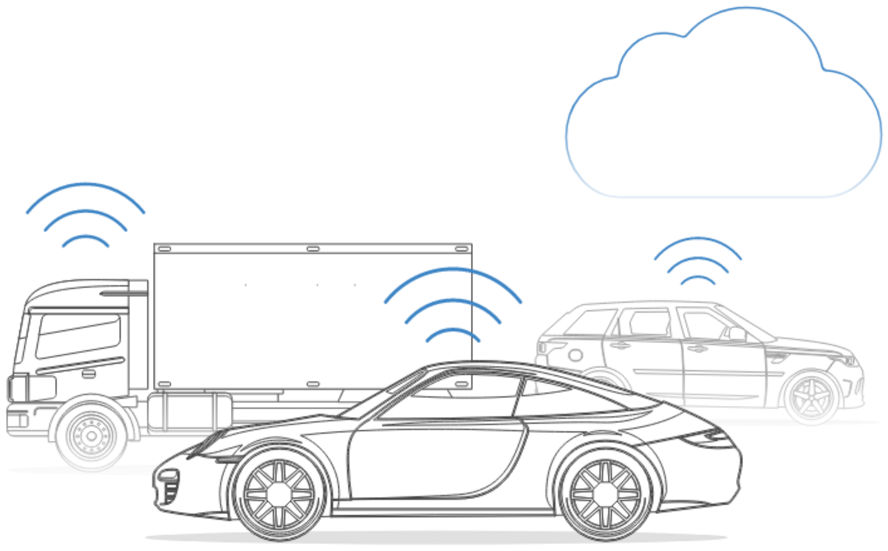
8) Under Switch Values option, select Roll_Down for Pressed and No_Request for Released.





Time	Chn	ID	Name	Event Type	Dir	DLC	Data
0.100442	CAN 1	0	EngineStatus	CAN Frame	Tx	2	00 00
0.200163	CAN 1	0	EngineStatus	CAN Frame	Tx	2	00 00
0.300892	CAN 1	0	EngineStatus	CAN Frame	Tx	2	00 00
0.400613	CAN 1	0	EngineStatus	CAN Frame	Tx	2	00 00
0.500361	CAN 1	0	EngineStatus	CAN Frame	Tx	2	00 00
0.600086	CAN 1	0	EngineStatus	CAN Frame	Tx	2	00 00
0.700815	CAN 1	0	EngineStatus	CAN Frame	Tx	2	00 00
0.800559	CAN 1	0	EngineStatus	CAN Frame	Tx	2	00 00
0.900275	CAN 1	0	EngineStatus	CAN Frame	Tx	2	00 00
1.001007	CAN 1	0	EngineStatus	CAN Frame	Tx	2	00 00
1.100751	CAN 1	0	EngineStatus	CAN Frame	Tx	2	00 00
1.200474	CAN 1	0	EngineStatus	CAN Frame	Tx	2	00 00
1.300206	CAN 1	0	EngineStatus	CAN Frame	Tx	2	00 00
1.400938	CAN 1	0	EngineStatus	CAN Frame	Tx	2	00 00
1.500673	CAN 1	0	EngineStatus	CAN Frame	Tx	2	00 00
1.600420	CAN 1	0	EngineStatus	CAN Frame	Tx	2	00 00
1.700146	CAN 1	0	EngineStatus	CAN Frame	Tx	2	00 00
1.800871	CAN 1	0	EngineStatus	CAN Frame	Tx	2	00 00
1.900600	CAN 1	0	EngineStatus	CAN Frame	Tx	2	00 00
2.000343	CAN 1	0	EngineStatus	CAN Frame	Tx	2	00 00
2.100097	CAN 1	0	EngineStatus	CAN Frame	Tx	2	00 00
2.200798	CAN 1	0	EngineStatus	CAN Frame	Tx	2	00 00
2.300544	CAN 1	0	EngineStatus	CAN Frame	Tx	2	00 00
2.400275	CAN 1	0	EngineStatus	CAN Frame	Tx	2	00 00

3-Chapter 3 : The Future of CAN-BUS :



Looking ahead, the CAN bus protocol will stay relevant - though it will be impacted by major trends:

- A need for increasingly advanced vehicle functionality
- The rise of cloud computing
- Growth in Internet of Things (IoT) and connected vehicles
- The impact of autonomous vehicles

In particular, the rise in connected vehicles (V2X) and cloud will lead to a rapid growth in vehicle telematics and IoT CAN loggers.