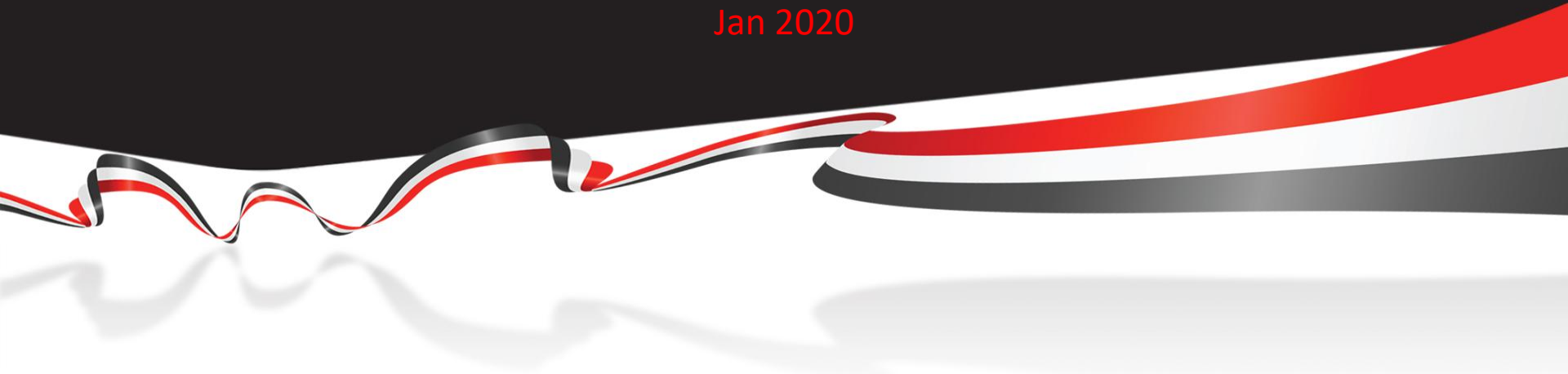




Custom CAN messaging with a DBC file

Jan 2020





Topics

- What is a DBC file
- Why use DBC Import
- Import a DBC File
- Transmit or Receive direction
- Initialise the CAN Bus
- Transmit messages
- Receive messages
- Multiplex messages
- Receive messages - Multiplex
- Transmit messages - Multiplex

Related content topics can be found in the M1 Build User Manual or the M1 Development User Manual in the Help menu of the M1 Build software



What is a DBC file?

- The DBC file is a CAN database file that describes the communication structure of a CAN network
- It's primary purpose is to describe how to translate raw CAN data within a CAN frame into human-readable signal values
- The DBC file defines nodes, messages and signals on a CAN network
 - Node: a transmitter or receiver of CAN communication on the network
 - Message: a block of data that contains multiple signals. Each message has a unique address which is used to determine the content
 - Signal: the actual data that is communicated to the CAN receiver, e.g. engine speed, throttle position
- The DBC format is proprietary to Vector Informatik GmbH and is the de-facto standard for the automotive industry
- DBC files can be created or administered with a CANdb editor
- DBC files can be read by CANalyzer or equivalent software



Why use DBC file import?

Importing a DBC file defines the structure of the CAN messages and signals and simplifies the CAN scripts.

The following settings are automatically imported:

- Message Address
- Message Length
- Signal Start Bit
- Signal Length
- Signal Byte Order (Motorola, Intel)
- Signal Value Type (Signed, Unsigned, IEEE float)
- Signal Factor
- Signal Offset
- Signal Unit

Only the message transmit frequency and the linked M1 channel need to be defined.

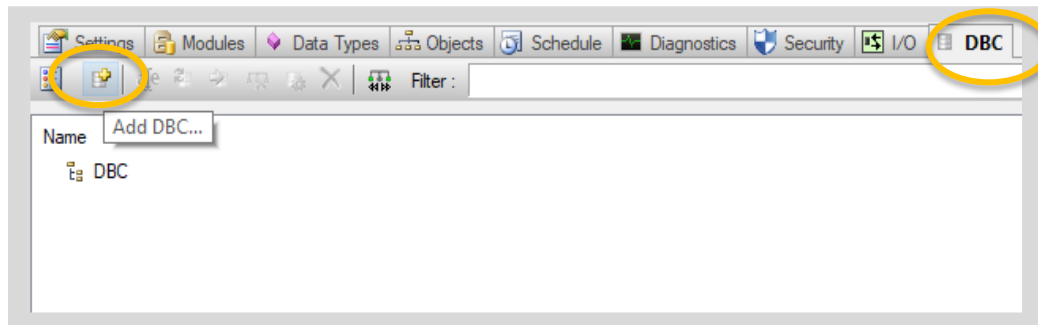


Import DBC file

1. Open M1 Build
2. Open or create a project
3. From the centre pane, select the **DBC** tab
4. Click the **Add DBC** icon
5. Enter a name that will be used to reference the DBC object (e.g. *Sample*) and click enter
6. A new window will pop up to navigate to the required DBC file
Note: This only happens when you have typed a new name.
If you used the default name, you can still import the file via **Edit > Import from file ...**
7. Click **Open** and all messages will be imported

Related content:

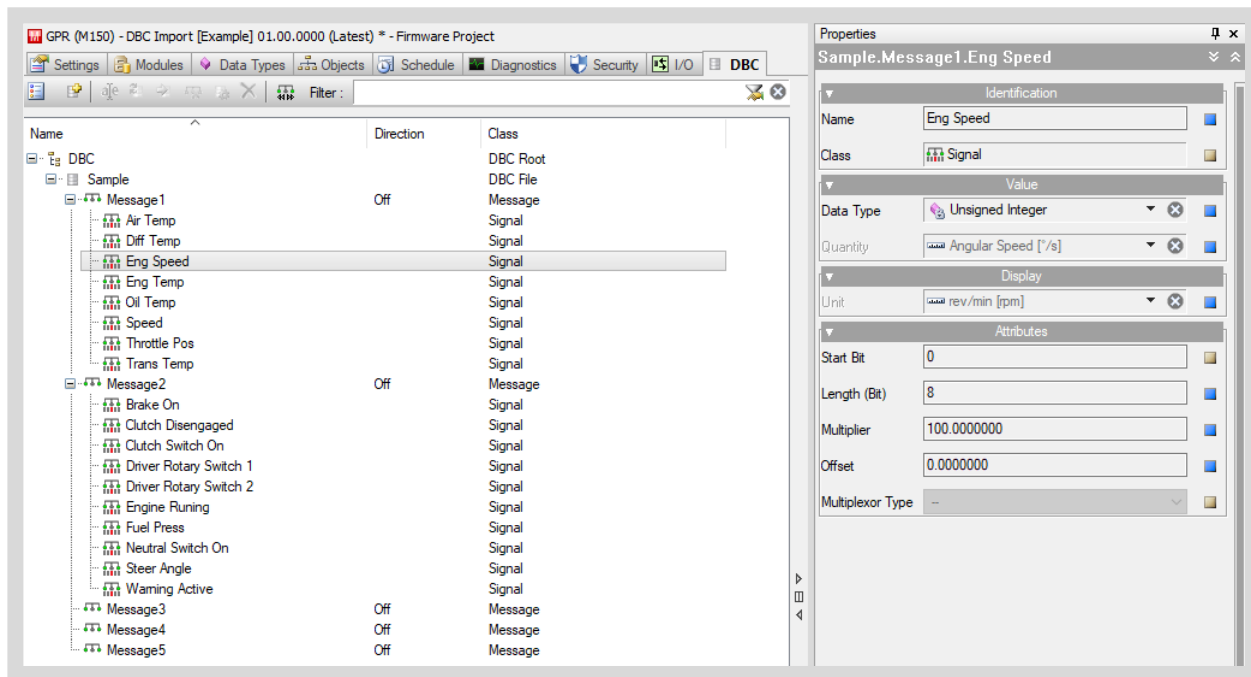
- Opening a Project





Import DBC file

All messages and signals contained in the DBC will be displayed and the imported structure of each message and signal is displayed in the properties window.



The screenshot shows the MoTeC M-Build software interface. The main window displays a tree view of the DBC file structure. The tree is organized as follows:

- DBC
 - Sample
 - Message1
 - Air Temp (Signal)
 - Diff Temp (Signal)
 - Eng Speed (Signal)
 - Eng Temp (Signal)
 - Oil Temp (Signal)
 - Speed (Signal)
 - Throttle Pos (Signal)
 - Trans Temp (Signal)
 - Message2
 - Brake On (Signal)
 - Clutch Disengaged (Signal)
 - Clutch Switch On (Signal)
 - Driver Rotary Switch 1 (Signal)
 - Driver Rotary Switch 2 (Signal)
 - Engine Running (Signal)
 - Fuel Press (Signal)
 - Neutral Switch On (Signal)
 - Steer Angle (Signal)
 - Warning Active (Signal)
 - Message3 (Message)
 - Message4 (Message)
 - Message5 (Message)

The Properties window on the right shows the details for the selected 'Eng Speed' signal:

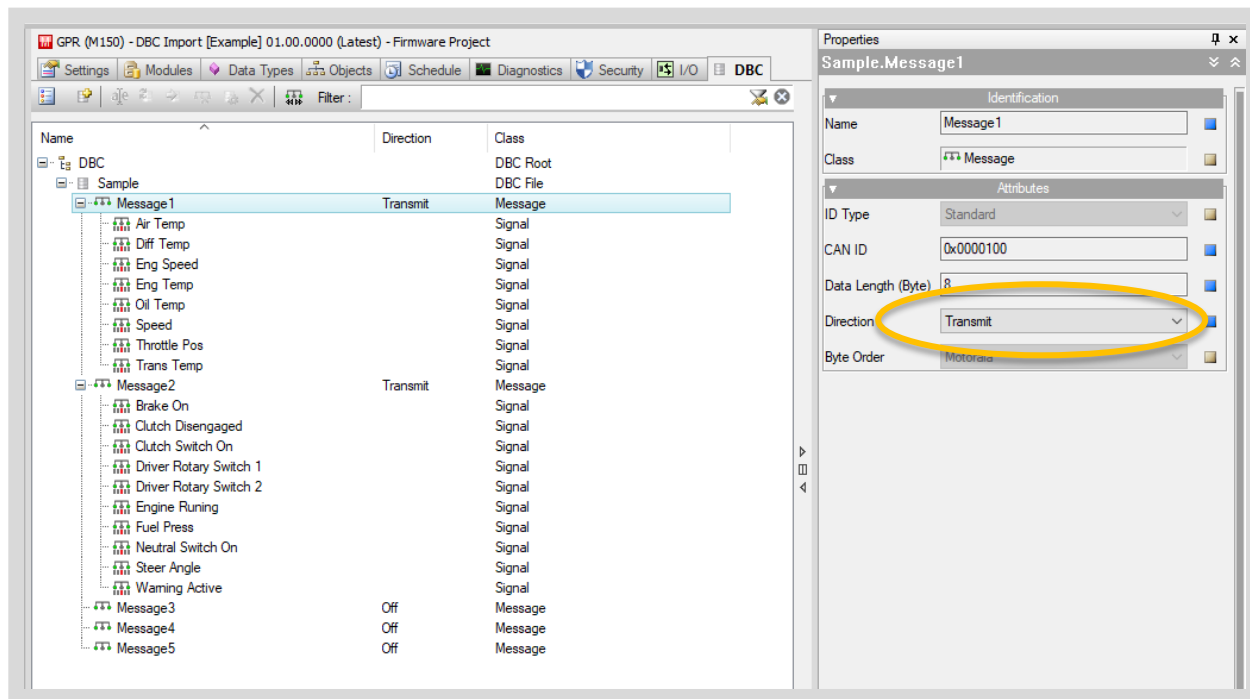
- Identification**
 - Name: Eng Speed
 - Class: Signal
- Value**
 - Data Type: Unsigned Integer
 - Quantity: Angular Speed [°/s]
- Display**
 - Unit: rev/min [rpm]
- Attributes**
 - Start Bit: 0
 - Length (Bit): 8
 - Multiplier: 100.0000000
 - Offset: 0.0000000
 - Multiplexor Type: --



Transmit or Receive

Set each message to be used as either Transmit or Receive.

1. Click the relevant message
2. In **Properties** (right pane), select **Direction** as either *Transmit* or *Receive*



The screenshot shows the MoTeC M-Build software interface. The main window displays a list of messages in a table with columns for Name, Direction, and Class. The 'Sample' folder is expanded, showing 'Message1' selected. The 'Properties' pane on the right shows the configuration for 'Sample.Message1'.

Name	Direction	Class
DBC		DBC Root
Sample		DBC File
Message1	Transmit	Message
Air Temp		Signal
Diff Temp		Signal
Eng Speed		Signal
Eng Temp		Signal
Oil Temp		Signal
Speed		Signal
Throttle Pos		Signal
Trans Temp		Signal
Message2	Transmit	Message
Brake On		Signal
Clutch Disengaged		Signal
Clutch Switch On		Signal
Driver Rotary Switch 1		Signal
Driver Rotary Switch 2		Signal
Engine Running		Signal
Fuel Press		Signal
Neutral Switch On		Signal
Steer Angle		Signal
Warning Active		Signal
Message3	Off	Message
Message4	Off	Message
Message5	Off	Message

The Properties pane for 'Sample.Message1' shows the following settings:

- Identification:**
 - Name: Message1
 - Class: Message
- Attributes:**
 - ID Type: Standard
 - CAN ID: 0x0000100
 - Data Length (Byte): 8
 - Direction: Transmit** (highlighted with a yellow circle)
 - Byte Order: Motorola



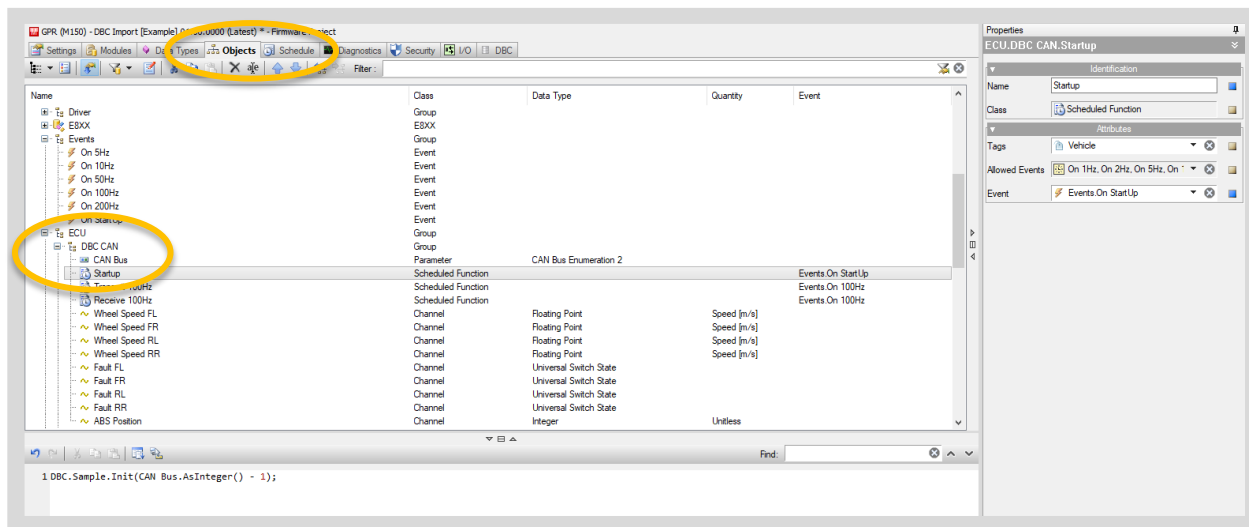
Initialise the CAN Bus

Initialising the CAN bus on startup, sets the CAN bus the M1 will use to transmit and receive the messages contained in the imported DBC file.

1. Select the **Objects** tab
2. Select or Add a **Group** and name it (e.g. DBC CAN)
3. Add a **Parameter** and name it (e.g. CAN Bus)

Related content:

- Add an Object to a Project
- Add code to a Function





Initialise the CAN Bus

4. Add a **Scheduled Function** and name it (e.g. Startup)
5. In **Properties**, set **Event** to *Events On StartUp*
Note: The Project needs to contain this **Event** class
6. Add the following code to this Scheduled Function
`DBC.Sample.Init(CAN Bus.AsInteger() - 1);`

Related content:

- Add Events
- Add an Object to a Project
- Add code to a Function

The screenshot shows the MoTeC M-Build software interface. The 'Objects' tab is active, displaying a tree view on the left and a table of objects on the right. The 'Startup' object is highlighted in the tree view and circled in yellow. The 'Properties' window on the right shows the 'Identification' tab for the 'Startup' object, with the 'Class' set to 'Scheduled Function' and the 'Event' set to 'Events On StartUp'. The 'Code' tab is also visible, showing the code snippet: `1 DBC.Sample.Init(CAN Bus.AsInteger() - 1);`, which is also circled in yellow.

Name	Class	Data Type	Quantity	Event
Driver	Group			
ESIX	Group			
Events	Group			
On 5Hz	Event			
On 10Hz	Event			
On 50Hz	Event			
On 100Hz	Event			
On 200Hz	Event			
On StartUp	Event			
ECU	Group			
DBC CAN	Group			
CAN Bus	Parameter	CAN Bus Enumeration 2		
Startup	Scheduled Function			Events On StartUp
Receive 100Hz	Scheduled Function			Events On 100Hz
Wheel Speed FL	Channel	Floating Point	Speed [m/s]	
Wheel Speed FR	Channel	Floating Point	Speed [m/s]	
Wheel Speed RL	Channel	Floating Point	Speed [m/s]	
Wheel Speed RR	Channel	Floating Point	Speed [m/s]	
Fault FL	Channel	Universal Switch State		
Fault FR	Channel	Universal Switch State		
Fault RL	Channel	Universal Switch State		
Fault RR	Channel	Universal Switch State		
ABS Position	Channel	Integer	Unless	

Transmit Messages

Each transmitted message needs to be scheduled at the required frequency.

In this example, the CAN Transmit uses 100 Hz event:

1. On the **Objects** tab, select the **Group** (*DBC CAN*)
2. Add a **Scheduled Function** and name it (e.g. *Transmit 100 Hz*)
3. In **Properties**, set **Event** to *Events on 100 Hz*
Note: The project needs to contain this **Event** class
4. Double click the *Transmit 100 Hz* **Scheduled Function** to access the code editor



Related content:

- Add Events
- Add an Object to a Project
- Add code to a Function



Transmit Messages

5. Add a Transmit Script to the code
 - Start each line with selecting or typing **Keyword DBC** followed by **Period(.)**
 - A suggestion box will show available next items
 - Help (left pane) will show relevant information about the options

This is an example of a complete Transmit code.

Note: Multiple Set methods are used depending on the signal type.

```

1 if (CAN Bus neq CAN Bus.Not in Use)
2 {
3     local ok = true;
4
5     local h = DBC.Sample.Message1.TxOpen() //handle which sets the byte order
6     DBC.Sample.Message1.TxInitialise(h); //Initialise Message1
7     DBC.Sample.Message1.Eng Speed.SetFromBaseUnit(h, Engine.Speed);
8     DBC.Sample.Message1.Throttle Pos.SetFromBaseUnit(h, Throttle.Position);
9     DBC.Sample.Message1.Speed.SetFromBaseUnit(h, Vehicle.Speed);
10    DBC.Sample.Message1.Eng Temp.SetFromBaseUnit(h, Coolant.Temperature);
11    DBC.Sample.Message1.Oil Temp.SetFromBaseUnit(h, Engine.Oil.Temperature);
12    DBC.Sample.Message1.Trans Temp.SetFromBaseUnit(h, Transmission.Temperature);
13    DBC.Sample.Message1.Diff Temp.SetFromBaseUnit(h, Differential.Temperature);
14    ok = DBC.Sample.Message1.Tx(h); //Transmit Message1
15
16    h = DBC.Sample.Message2.TxOpen() //handle which sets the byte order
17    DBC.Sample.Message2.TxInitialise(h); //Initialise Message2
18    DBC.Sample.Message2.Fuel Press.SetFromBaseUnit(h, Fuel.Pressure);
19    DBC.Sample.Message2.Steer Angle.SetFromBaseUnit(h, Steering.Angle);
20    DBC.Sample.Message2.Engine Running.SetBit(h, Engine.State eq Engine.State.Run);
21    DBC.Sample.Message2.Warning Active.SetBit(h, Warning.Source neq Warning.Source.None);
22    DBC.Sample.Message2.Brake On.SetBit(h, Brake.State eq Brake.State.On);
23    DBC.Sample.Message2.Neutral Switch On.SetBit(h, Gear.Neutral Switch eq Gear.Neutral Switch.On);
24    DBC.Sample.Message2.Clutch Switch On.SetBit(h, Clutch.Switch eq Clutch.Switch.On);
25    DBC.Sample.Message2.Clutch Disengaged.SetBit(h, Clutch.State eq Clutch.State.Disengaged);
26    DBC.Sample.Message2.Driver Rotary Switch 1.SetInteger(h, Driver.Rotary Switch 1.AsInteger());
27    DBC.Sample.Message2.Driver Rotary Switch 2.SetInteger(h, Driver.Rotary Switch 2.AsInteger());
28    ok = DBC.Sample.Message2.Tx(h); //Transmit Message2
29
30    ok = ok;
31 }
    
```



Transmit Messages

A Transmit Script contains the following:

- Opening the transmit handle: **TxOpen()**
- Message initialise function: **TxInitialise()**
- Signal set function: There are multiple functions that can be used to set the values of the CAN message signals.
 - **SetFromBaseUnit()**: converts the given value into the signal's units before applying the required signal scale and offset
 - **SetScaled()**: applies the signal's scale and offset to the given value
 - **SetBit()**: when the condition is true the signal value is set to 1 otherwise the signal is set to 0.
 - **SetUnsignedInteger()**: populates the signal value as an unsigned integer with no scale or offset. The given value must be, or converted to be, an unsigned integer.
 - **SetInteger()**: populates the signal value as an integer with no scale or offset. The given value must be, or converted to be, an integer.
- Message transmit function: **Tx()** transmits the message with all the values that were assigned by the set functions.

All functions take the CAN transmit handle argument.



Receive Messages

Each received message needs to be scheduled at the required frequency.

In this example for an ABS unit the CAN Receive uses a 100 Hz event:

1. On the **Objects** tab, select the **Group** (*DBC CAN*)
2. Add a **Scheduled Function** and name it (e.g. *Receive 100 Hz*)
3. In **Properties**, set **Event** to *Events on 100 Hz*
Note: The project needs to contain this **Event** class
4. Double click the *Receive 100 Hz* **Scheduled Function** to access the code editor

Related content:

- Add Events
- Add an Object to a Project
- Add code to a Function



Receive Messages

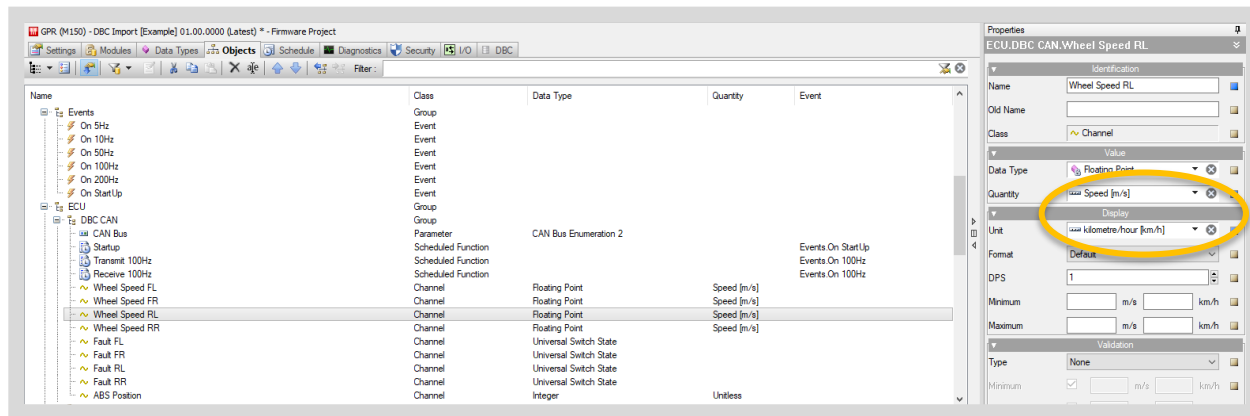
5. Add required **Channels** and name them.
6. In **Properties**, set **Quantity** and **Unit**

Related content:

- Add an Object to a Project

Channels for the ABS example:

- four wheel speed channels;
Wheel Speed FL, Wheel Speed FR, Wheel Speed RL and Wheel Speed RR
Quantity set to *Speed*, **Unit** set to *kilometre/hour*
- four fault channels;
Fault FL, Fault FR, Fault RL and Fault RR
Quantity set to *Universal Switch State*
- a position channel;
ABS Position
Quantity set to *Integer*





Receive Messages

7. Add a Receive Script to the code

- Start each line with selecting or typing **Keyword** *DBC* followed by **Period(.)**
- A suggestion box will show available next items
- Help (left pane) will show relevant information about the options

This is an example of a complete Receive Script.

Note: Multiple Get methods are used depending on the signal type.

```
1 /* Local variables for time (in seconds) after which the message will time out.
2 Suggested value 10 times the message frequency */
3 static local hrtmessage3 = 0.1;
4 static local hrtmessage4 = 0.5;
5
6 if (CAN Bus neq CAN Bus.Not in Use)
7 {
8     if (DBC.Sample.Message3.Receive()) //Message3 Received
9     {
10         Wheel Speed FL = DBC.Sample.Message3.ABS Wheel Speed FL.GetToBaseUnit();
11         Wheel Speed FR = DBC.Sample.Message3.ABS Wheel Speed FR.GetToBaseUnit();
12         Wheel Speed RL = DBC.Sample.Message3.ABS Wheel Speed RL.GetToBaseUnit();
13         Wheel Speed RR = DBC.Sample.Message3.ABS Wheel Speed RR.GetToBaseUnit();
14         Fault FL = (DBC.Sample.Message3.ABS Fault FL.GetBit()) ? Fault FL.On : Fault FL.Off;
15         Fault FR = (DBC.Sample.Message3.ABS Fault FR.GetBit()) ? Fault FR.On : Fault FR.Off;
16         Fault RL = (DBC.Sample.Message3.ABS Fault RL.GetBit()) ? Fault RL.On : Fault RL.Off;
17         Fault RR = (DBC.Sample.Message3.ABS Fault RR.GetBit()) ? Fault RR.On : Fault RR.Off;
18         hrtmessage3 = 0.1; //Reset to initial value
19     }
20     else if (hrtmessage3 > 0.0) //Counts down when message 3 is not received
21     {
22         hrtmessage3 = hrtmessage3 - Receive 100Hz.Period();
23     }
24     else //Message3 Not Received
25     {
26         Wheel Speed FL = Calculate.NAN();
27         Wheel Speed FR = Calculate.NAN();
28         Wheel Speed RL = Calculate.NAN();
29         Wheel Speed RR = Calculate.NAN();
30         Fault FL = Fault FL.On;
31         Fault FR = Fault FR.On;
32         Fault RL = Fault RL.On;
33         Fault RR = Fault RR.On;
34     }
35
36     if (DBC.Sample.Message4.Receive()) //Message4 Received
37     {
38         ABS Position.Set(DBC.Sample.Message4.ABS Position.GetInteger());
39         hrtmessage4 = 0.5; //Reset to initial value
40     }
41     else if (hrtmessage4 > 0) //Counts down when message 4 is not received
42     {
43         hrtmessage4 = hrtmessage4 - Receive 100Hz.Period();
44     }
45     else //Message4 Not Received
46     {
47         ABS Position = 0;
48     }
49 }
```



Receive Messages

The Receive Script contains for each message:

- A local variable used to count down a time out if the message is not received
- Receive function: **Receive()** returns a true value if the message is received
- Get function: There are multiple functions that can be used to get the values from the CAN message signals.
 - **GetFromBaseUnit()**: Applies the required scale and offset to the signal and then converts from the signal's units to the M1 build base units.
 - **GetScaled()**: Applies the required scale and offset to the signal.
 - **GetBit()**: true when the signal value is 1 otherwise false.
 - **GetUnsignedInteger()**: extracts the signal value as an Unsigned Integer with no scale or offset.
 - **GetInteger()**: extracts the signal value as an Integer with no scale or offset.



Multiplex Messages

The multiplex format allows for more data to be transmitted on a single message ID.

A Multiplexor signal, transmitted with each message, determines which signals are available in each message.

This example will use the DBC file for a MoTeC LTC (Lambda to CAN) module.

The LTC transmits multiple data sets on the same message ID using the multiplex format with LTC1_Index as the Multiplexor Signal.

Message LTC_1_01 (0x480)

Definition Signals Transmitters Receivers Layout Attributes Comment

Multiplexor Signal: LTC1_Index = 0x0

	0	1	2	3	4	5	6	7
0	LTC1_Index							
1	_msh	9	1	2	3	4	5	6
2	LTC1_Lambda							
3	_msh	9	9	10	11	12	13	14
4	LTC1_Lambda							
5	LTC1_Bn							
6	_msh	24	25	26	27	28	29	30
7	LTC1_Bn							
8	LTC1_InternalTemp							
9	_msh	40	41	42	43	44	45	46
10	LTC1_SensorCool							
11	LTC1_SensorIntake							
12	LTC1_SensorIn							
13	LTC1_HeaterFan							
14	LTC1_HeaterOp							
15	LTC1_HeaterShut							
16	LTC1_HeaterDutyCycle							
17	_msh	56	57	58	59	60	61	62

Arrange To Front To Back Add Remove BT Index Inverted

OK Cancel Apply Help

Multiplexor = 0x0

Message LTC_1_01 (0x480)

Definition Signals Transmitters Receivers Layout Attributes Comment

Multiplexor Signal: LTC1_Index = 0x1

	0	1	2	3	4	5	6	7
0	LTC1_Index							
1	_msh	0	1	2	3	4	5	6
2	LTC1_BattVots							
3	_msh	16	17	18	19	20	21	22
4	LTC1_Bn							
5	_msh	32	33	34	35	36	37	38
6	LTC1_Bn							
7	LTC1_RI							
8	_msh	40	49	50	51	52	53	54
9	LTC1_RI							
10	_msh	56	57	58	59	60	61	62

Arrange To Front To Back Add Remove BT Index Inverted

OK Cancel Apply Help

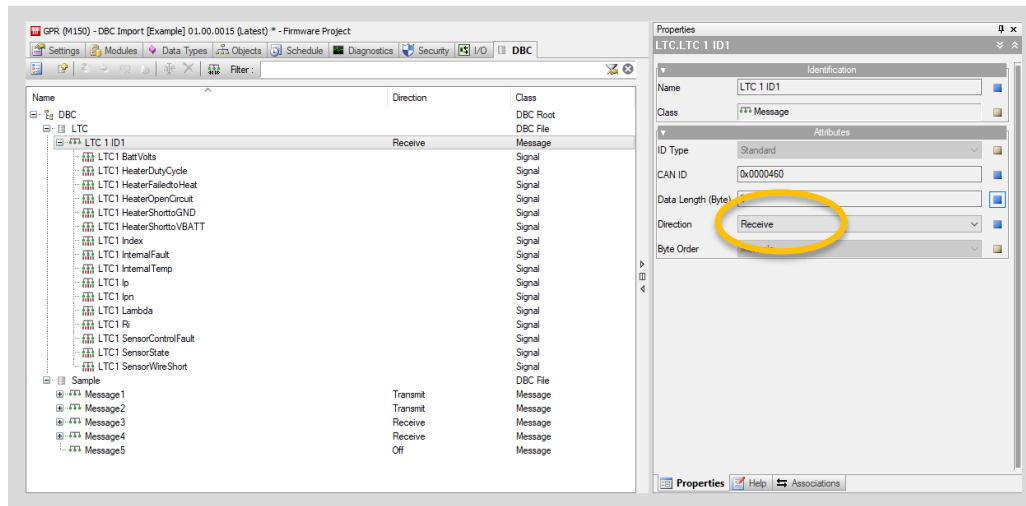
Multiplexor = 0x1



Receive Messages - Multiplex

Similar to the steps outlined in Import DBC file:

1. Add a new **DBC** object and name it (e.g. *LTC*)
2. Import the MoTeC_LTC_Rev2.dbc file.
3. Click to select message LTC 1 ID1
4. In **Properties** (right pane), set **Direction** to *Receive*





Receive Messages - Multiplex

Similar to steps outlined in setting up regular Receive Messages:

1. On the **Objects** tab, create a new **Group** and name it (e.g. *LTC CAN*)
2. Add a **CAN Bus Parameter**, and *Receive 100Hz* and **Startup Scheduled Function**
Tip: you can copy these objects from the *DBC CAN* group, paste them into this new group and adjust as required
3. Add required **Channels** and name them
In **Properties**, set **Quantity** and **Unit**

Channels for the LTC example:

- **Channel** *Lambda*
Quantity set to *Air fuel Ratio [LA]*
- **Channel** *Internal Temperature*
Quantity set to *Temperature [°C]*
- **Channel** *Battery Voltage*
Quantity set to *Voltage [V]*

Related content:

- Add an Object to a Project



Receive Messages - Multiplex

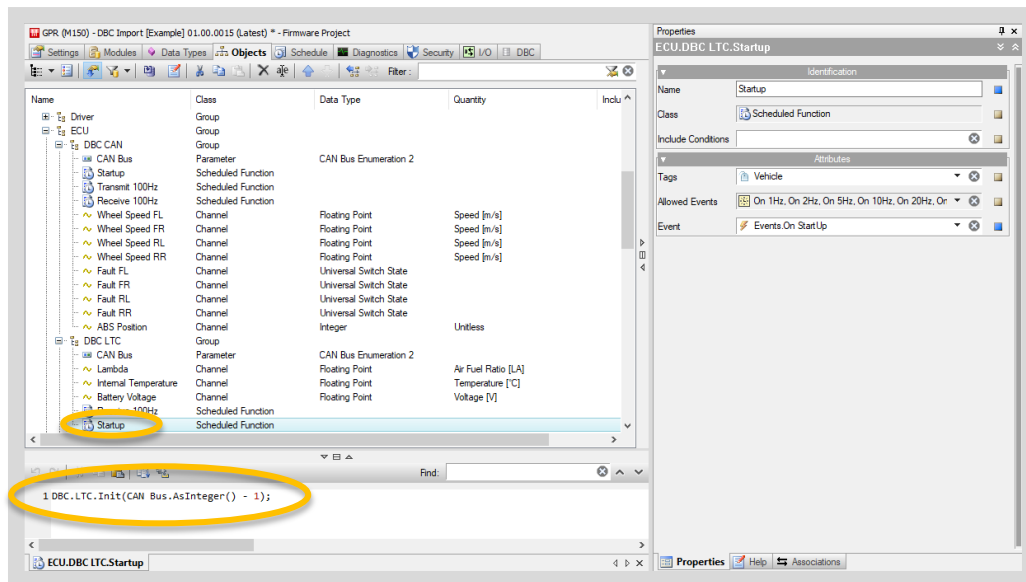
To link the DBC to the required CAN bus as determined by **ECU.DBC LTC.CAN Bus:**

Related content:

- Add code to a Function

1. Open the *Startup* Scheduled Function and add the following code to this Scheduled Function

```
DBC.LTC.Init(CAN_Bus.AsInteger()-1);
```





Receive Messages - Multiplex

This is an example of a complete Multiplex Receive script. Add this Receive Script to the *Receive 100 Hz* Scheduled Function

The Multiplex Receive script is similar to the regular Receive script with the following differences:

1. **RxFillBuffer ()**: fills the buffer with up to 32 messages received since the previous calculation cycle
2. **FindMessage (multiplex value)**: finds the newest message that matches the multiplexor value
3. Use various **Get** functions to retrieve the signal values from each message

```
1 static local hrto460 = 0.1; //Timeout variable
2
3 if (CAN Bus neq CAN Bus.Not in Use)
4 {
5     /*
6     * 0x0460 - LTC - 100hz receive
7     */
8     DBC.LTC.LTC 1 ID1.RxFillBuffer(); //Fills the CAN Rx Buffer
9
10    if (DBC.LTC.LTC 1 ID1.FindMessage(0x0) //Message 0x460 Index 0 Received
11    {
12        Lambda = DBC.LTC.LTC 1 ID1.LTC1 Lambda.GetToBaseUnit();
13        Internal Temperature = DBC.LTC.LTC 1 ID1.LTC1 InternalTemp.GetToBaseUnit();
14        if (DBC.LTC.LTC 1 ID1.FindMessage(0x1) //Message 0x460 Index 1 Received
15        {
16            Battery Voltage = DBC.LTC.LTC 1 ID1.LTC1 BattVolts.GetToBaseUnit();
17        }
18        hrto460 = 0.1; //Reset Timeout variable
19    }
20    else if (hrto460 > 0.0) //Delay Time Out if Message Index 0 not Received
21    {
22        hrto460 = hrto460 - Receive 100Hz.Period();
23    }
24    else //Message Index 0 Timed Out
25    {
26        Lambda = Calculate.NAN();
27        Internal Temperature = Calculate.NAN();
28        Battery Voltage = Calculate.NAN();
29    }
30 }
```



Transmit Messages - Multiplex

Messages can also be transmitted as multiplex by following the steps in the [Transmit Messages](#) section.

This example shows how to transmit the LTC multiplex message using the same message as the [Receive Messages - Multiplex](#) section.

This is a complete Multiplex Transmit script. Add this Transmit script to a *Transmit 100 Hz Scheduled Function*

The Multiplex Transmit script is similar to the regular Transmit script with the following differences:

1. Set the Multiplexor signal on the LTC1 Index signal with **SetUnsignedInteger (h, multiplex value())**
2. Use various **Set** functions as required

```
1 if (CAN Bus neq CAN Bus.Not in Use)
2 {
3     local ok = true;
4
5     local h = DBC.LTC.LTC 1 ID1.TxOpen(); //handle which sets the byte order
6     DBC.LTC.LTC 1 ID1.TxInitialise(h); //Initialise Message
7     DBC.LTC.LTC 1 ID1.LTC1 Index.SetUnsignedInteger(h, 0x0); //Set Multiplexor Signal
8     //Set all channels when Multiplexor Signal is 0x0
9     DBC.LTC.LTC 1 ID1.LTC1 Lambda.SetFromBaseUnit(h, Lambda);
10    DBC.LTC.LTC 1 ID1.LTC1 InternalTemp.SetFromBaseUnit(h, Internal Temperature);
11    ok = DBC.LTC.LTC 1 ID1.Tx(h); //Transmit Message
12
13    DBC.LTC.LTC 1 ID1.TxInitialise(h); //Initialise Message
14    DBC.LTC.LTC 1 ID1.LTC1 Index.SetUnsignedInteger(h, 0x1); //Set Multiplexor Signal
15    //Set all channels when Multiplexor Signal is 0x1
16    DBC.LTC.LTC 1 ID1.LTC1 BattVolts.SetFromBaseUnit(h, Battery Voltage);
17    ok = DBC.LTC.LTC 1 ID1.Tx(h); //Transmit Message
18
19    ok = ok;
20 }
```