TECH NOTE :: QuantumX and SAE J1939 CAN bus

Version: 2015-01-30
Author: Christof Salcher, Product Manager Test & Measurement, HBM Germany
Status: public

Abstract

This Tech Note describes how to acquire SAE J1939 CAN signals with QuantumX MX840A or MX471 and the PC software catmanEasy from HBM.

Intro

QuantumX is a modular distributable data acquisition solution from HBM for measurement and testing solving demanding engineering tasks for quicker innovation. The data acquisition modules offer highly accurate inputs acquiring physical quantities in the wide field of **mechanical, hydraulics, thermal** and **electrical or mixed systems** with data rates from 0.1 to 100 kS/sec and channel. QuantumX acquires sensor or transducer inputs measuring **force, strain, torque, pressure, temperature, displacement, speed, position, acceleration, flow, voltage, current and many more**. QuantumX offers superb A/D inputs supporting **voltage, current, bridge based inputs for strain gage or inductive transducers, LVDT, resistive, thermocouple, digital pulses** and signals from data busses like **CAN bus** parallel and time synchronous to all other inputs.
Some modules have output capability and work as signal converter (input to output), signal generator, real-time reaction (analog, digital) or gateway to other digital busses then Ethernet or FireWire like CAN bus or EtherCAT bus.

SAE J1939 Data Link Layer and QuantumX hardware

The J1939 protocol requires a CAN controller supporting active CAN2.0B. QuantumX offers CAN bus in and outputs. Both available units are equipped with an ISO11898 transceiver (Philips PCA82C251) supporting CAN2.0B. This transceiver is compliant to SAE J1939 protocol and can be used in 12 V or 24 V systems.

QuantumX module MX471 is a 4 channel CAN Module and MX840A is universal amplifier offering one channel with a compatible controller. CAN messages with standard 11 Bit identifier as well as messages with extended 29 bit identifier can be sent and received. Different CAN baud rate can be set. The CAN protocol uses the identifier to prioritize the bus access and to identify the messages. QuantumX does not support the “Transport Protocol Transfer Sequences” (TPTS) as it is described in the SAE Standard SAE 1939/21 Appendix C. TPTS are used for the transfer of data arrays of more than 8 byte length because a CAN message is limited to 8 byte.

HBMs software catmanEASY allows you to parameterize the CAN ports over an integrated J1939 CAN database file (dbc file). Some screenshots of the software can be found at the very end of this TECH NOTE.

In case you want to listen to certain J1939 messages using different software you can always set up certain messages calculating the PGN into the identifier in a manual way.

Constraints:

- Please read the datasheet of the corresponding module, for example MX471 or MX840A
- MX471 has an internal termination resistor with 120 Ohm which can terminate the CAN bus via software control.
- Only one signal value per CAN ID can be acquired (not multiple signals in one message)
- All acquired signals are transferred to FLOAT value (4 Byte) on FireWire or Ethernet, so it is not possible to acquire a signal on channel one in integer format and send it out on channel no 2 in the same format
- Sending CAN bus signals as analog voltage output is possible over FireWire and modules like MX878 or MX879 is possible
- MX471 has

catmanEASY – automatic parameterization of a CAN port listen to J1939 messages

catman only allows to parameterize receiving CAN signals. QuantumX Assistant allows you in addition a mapping of input to CAN bus.

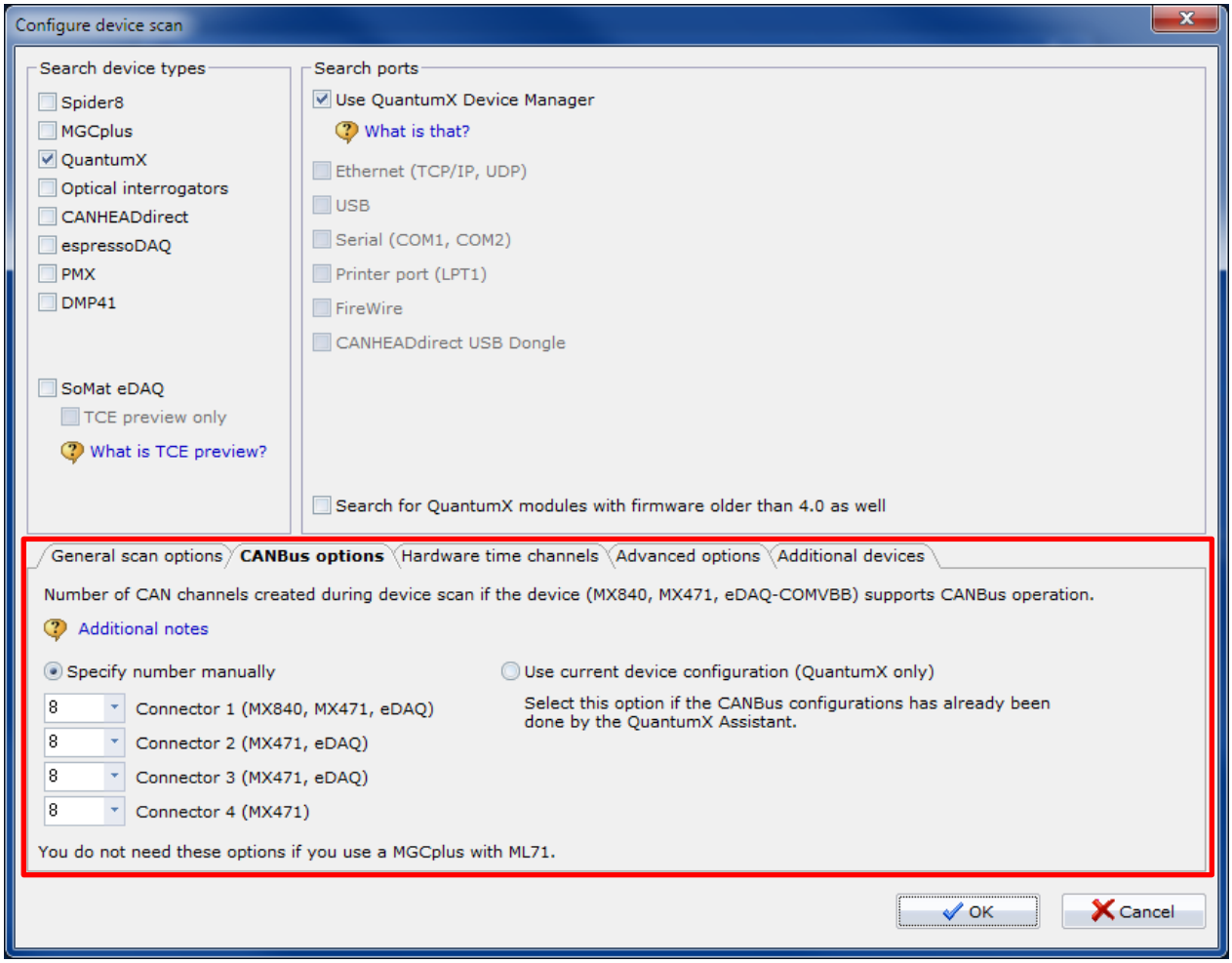
In contrast to an analog input connector which corresponds to exactly one channel in catman, a CAN connector provides a multitude of channels. To every channel you will later assign a CAN signal (i.e. a sensor describing the signal's parameters like message ID, frame format, start bit etc.) or by using a corresponding database file in format DBC.

During a device scan catman needs to know the amount of signals created for the specific CAN port. This determines the maximum number of CAN signals you can acquire. This number should roughly match the number of CAN signals you expect to acquire in your measurement project - creating a large number of channels which later on are never used will unnecessarily slow down the system and increase memory consumption.

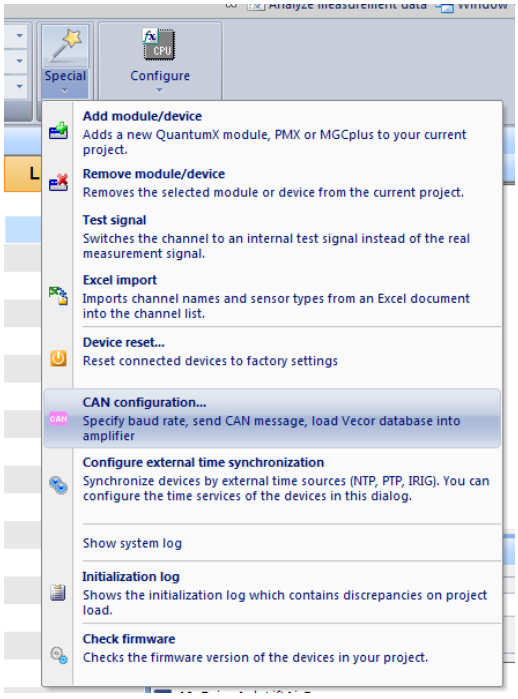
Switching port 1 of MX840A from analog mode to CANbus can be easily done by dragging a CAN signal to the first channel. catman will then prompt for the number of CAN channels to be created.

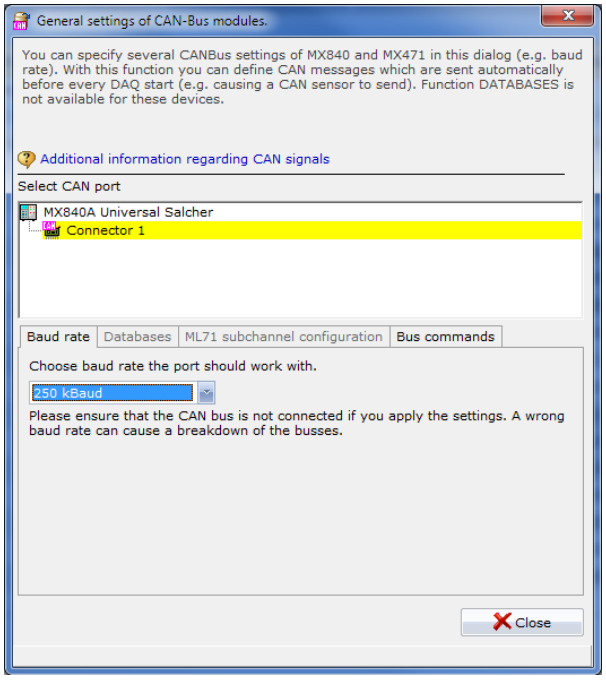
The maximum number of CAN signals per port is 128.

Now start catmanEasy and configure device scan. Go to tab CANBus options.
Here you can specify the amount of CAN signals you want to acquire per port or you can read out an existing device configuration out of the device.

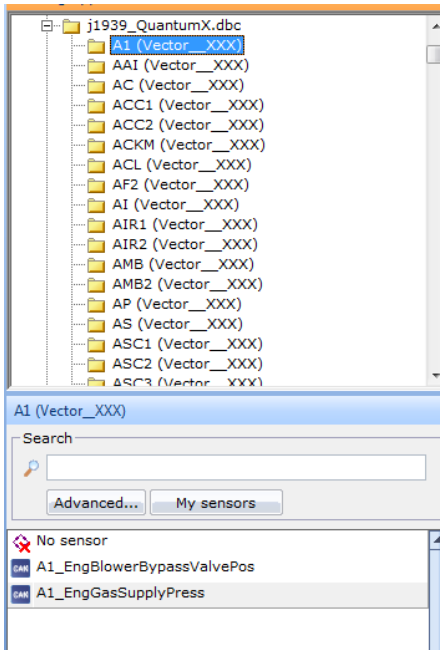


Start a new or an existing project with the necessary QuantumX modules.
Configure the CAN ports: Special -> CAN configuration





Open the sensor database in the DAQ channel list and drag and drop the signal to be acquired from the database:



Check the channels

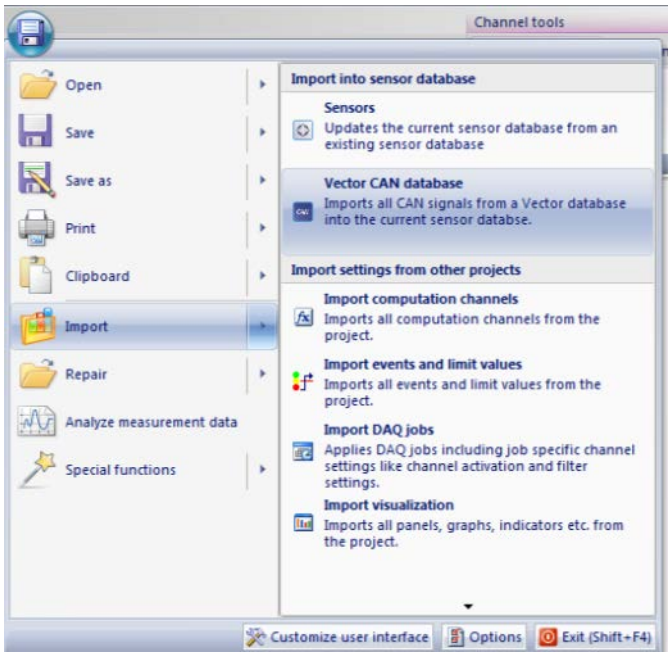
-	Channel name	Sensor/Function
	MX840A Universal Salcher [MX840A] [UUID=9E500281B] [Sync-Single] [172.2	
	A1_EngBlowerBypassValvePos	A1_EngBlowerBypassValvePos (Port: 1)
	AF2_EngGasFuelCorrectionFactor	AF2_EngGasFuelCorrectionFactor (Port:
	AI_DriveAxleLiftAirPress	AI_DriveAxleLiftAirPress (Port: 1)
	CAN 4	No CAN sensor assigned
	CAN 5	No CAN sensor assigned
	CAN 6	No CAN sensor assigned
	CAN 7	No CAN sensor assigned
	CAN 8	No CAN sensor assigned

Start your DAQ job.

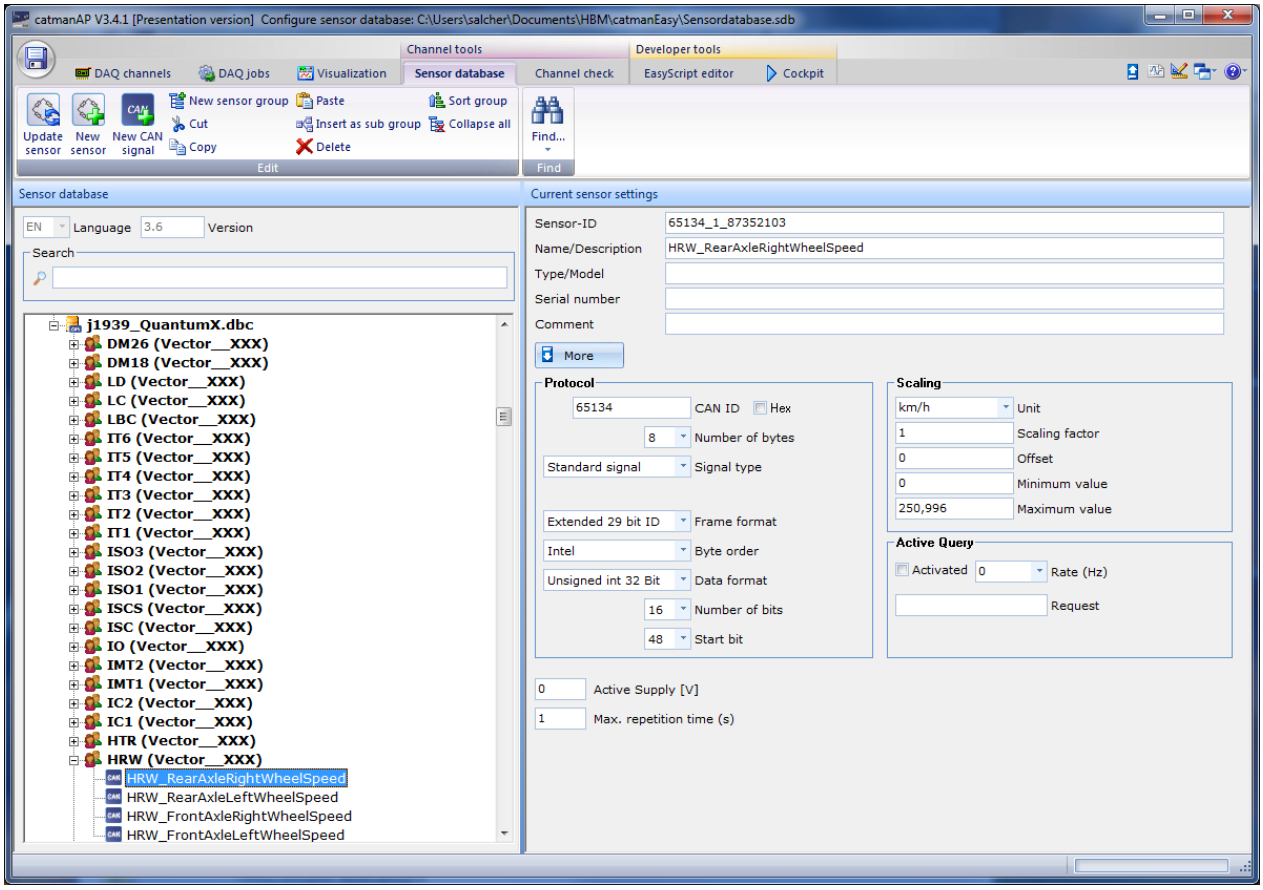
catmanEasy - manual parameterization of a CAN port listen to J1939 messages

The following example shows a configuration with PID EEC1, engine speed entered in the CAN database in catmanEASY.

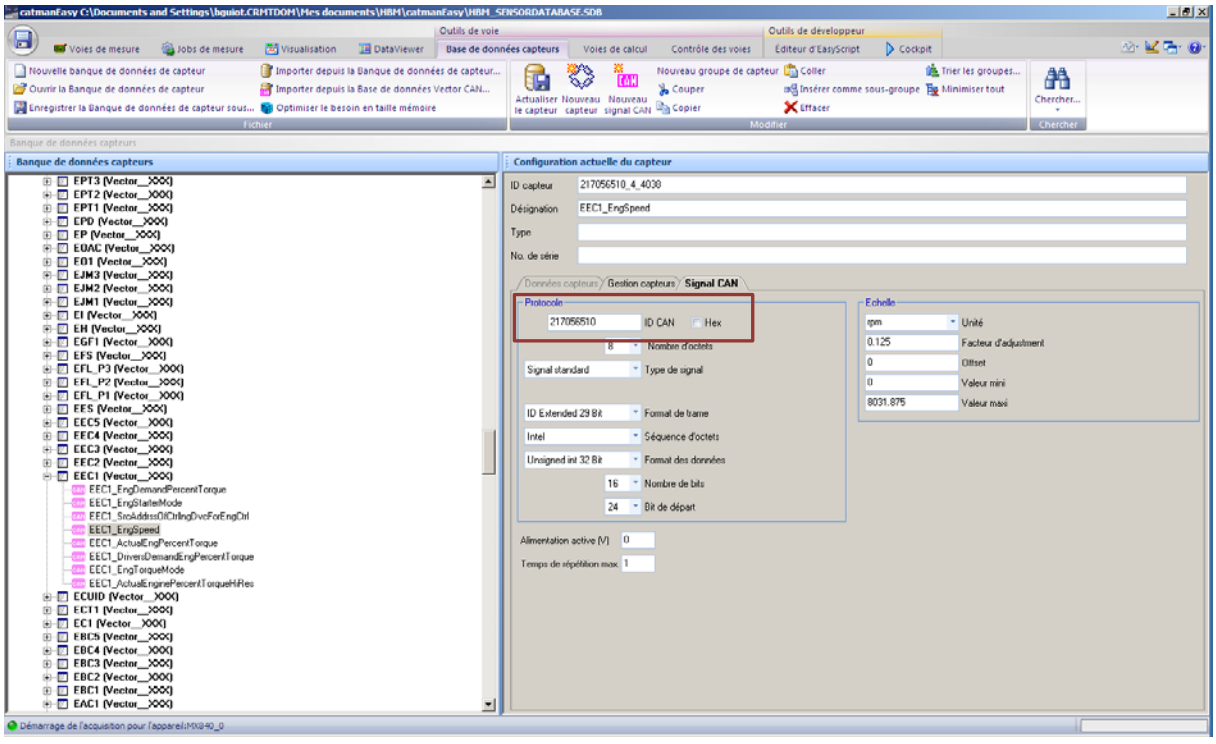
Read in a different J1939 DBC files into catmanEASY:



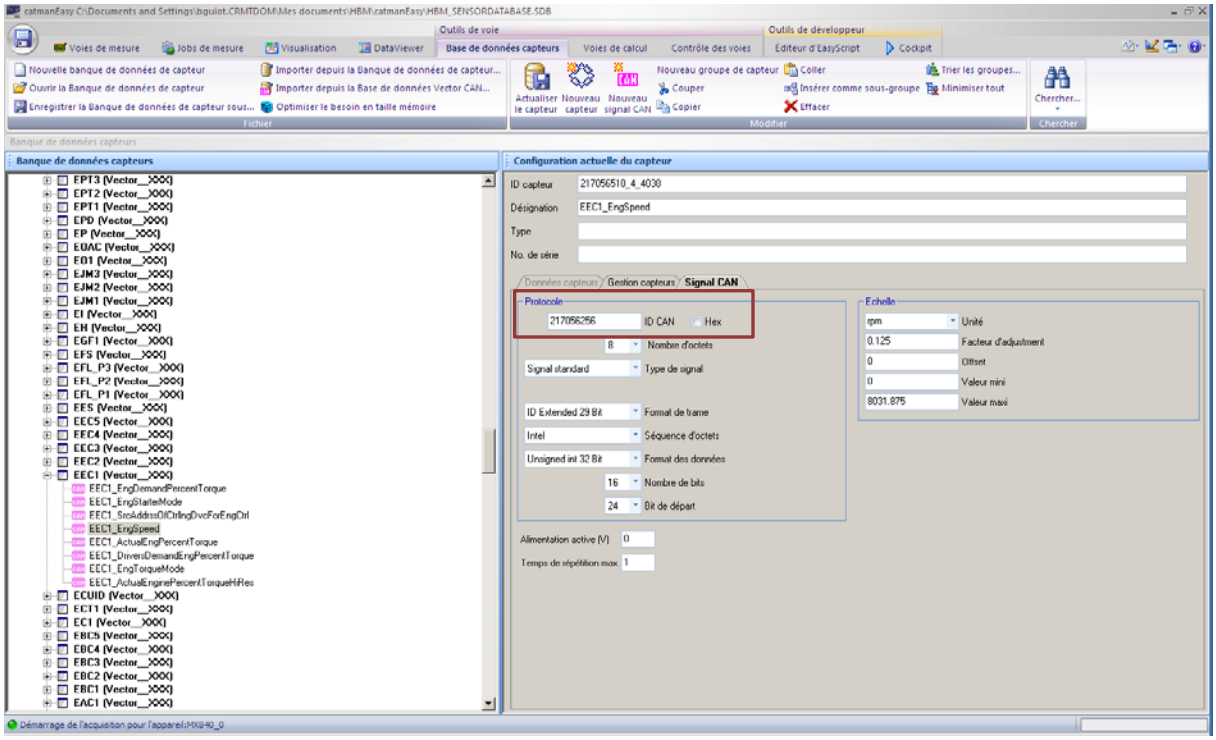
View and modify DBC file in catmanEASY database:



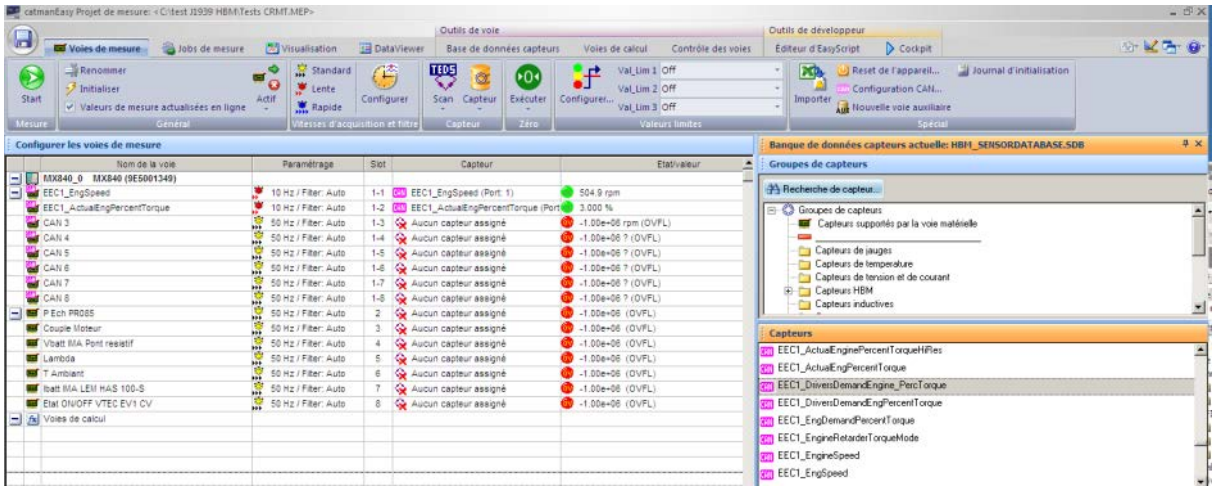
Original CAN-ID 217056**510**



After modification of CAN-ID to 217056**256**



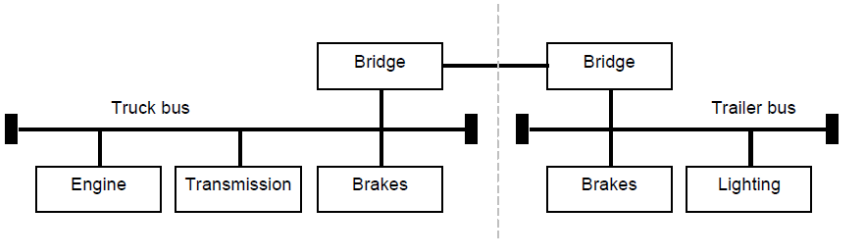
Start Measurement : 504.9 rpm



General information about SAE J1939

J1939 is a set of standards defined by SAE. They are used in heavy-duty vehicles – for example trucks and buses. In commercial vehicles different system parts like engine or trailer have their own electronic control units (ecu) which are connected to each other over this digital communication bus based on CAN technology so that all parts can be easily controlled. J1939 is a higher CAN protocol which sends broadcast messages to all participants like the standard CAN and to specific participants.

The physical layer (J1939/11) describes the electrical interface to the bus. The data link layer (J1939/21) describes the rules for constructing a message, accessing the bus, and detecting transmission errors. The application layer (J1939/71 and J1939/73) defines the specific data contained within each message sent across the network.



The particular characteristics of J1939 are:

- 29-bit identifier
- In most of the cases speed is 250 kBit/s
- Peer-to-peer and broadcast communication
- Transport protocols for up to 1785 data bytes
- Network management
- Definition of parameter groups

Parameter Groups (PGN) with 8 bytes in length can be created and send as a single message. The 8 bytes have to be packed out in the receiver unit.

Most messages defined by the J1939 standard are intended to be broadcast. This means that the data is transmitted on the network without a specific destination. This permits any device to use the data without requiring additional request messages. This also allows future software revisions to easily accommodate new devices (address assignments). When a message must be directed to a particular device, a specific destination address can be included within the message identifier. For example, a request for a specific torque value from the engine instead of a specific torque value from the brake controller.

The identifier is used slightly different in a message with a destination address (”PDU 1”) compared to a message intended for broadcast (”PDU 2”).

PDU stands for Protocol Data Unit (i.e. Message Format).

The SOF, SRR, and IDE bits are defined by the CAN standard and will be ignored here. The RTR bit (remote request bit) is always set to zero in J1939.

The 29-bit identifier used in J1939 is structured in the following way.

Priority	Reserved	Data page	PDU format	PDU specific	Source Address
3 bits	1 bit	1 bit	8 bits	8 bits	8 bits

The first three bits of the identifier are used for controlling a message’s priority during the arbitration process. A value of 0 has the highest priority. Higher priority values are typically given to high-speed control messages, for example, the torque control message from the transmission to the engine. Messages containing data that is not time critical, like the vehicle road speed, are given lower priority values.

The next bit of the identifier is reserved for future use and should be set to 0 for transmitted messages.

The next bit in the identifier is the data page selector. This bit expands the number of possible Parameter Groups that can be represented by the identifier.

The PDU format (PF) determines whether the message can be transmitted with a destination address or if the message is always transmitted as a broadcast message.

The interpretation of the PDU specific (PS) field changes based on the PF value:

If the PF is between 0 and 239, the message is addressable (PDU1) and the PS field contains the destination address.
If the PF is between 240 and 255, the message can only be broadcast (PDU2) and the PS field contains a Group Extension.
The Group extension expands the number of possible broadcast Parameter Groups that can be represented by the identifier.

The term Parameter Group Number (PGN) is used to refer to the value of the Reserve bit, DP, PF, and PS fields combined into a single 18 bit value.

Example: The ID 0xCF004EE can be divided into the following fields in the next table:

Priority				**Reserved**	**Data page**	**PDU format**
000	011	0	0	11110000	00000100	11101110
---	Prio	R	DP	PF	PS	SA

- PGN = the R, DP, PF and PS fields - in this case 0x0F004.
- PF = 0xF0 = 240, i.e. this is a PDU2 (broadcast) message
- PS = 0x04, i.e. the Group Extension = 4

The last 8 bits of the identifier contain the address of the device transmitting the message. The address is the label or "handle" which is assigned to provide a way to uniquely access a given device on the network. For a given network, every address must be unique (254 available). This means that two different devices (ECUs) cannot use the same address.

Addresses and Names (J1939/81)
The Name is a 64 bit (8 bytes) long label which gives every ECU a unique identity. The Name is composed of 10 fields and has the following structure shown in table 3.

Structure of the Name

1. Arbitrary address bit
2. Industry group, length 3 bits
3. Vehicle system instance, length 4 bits
4. Vehicle system, length 7 bits
5. Reserved bit
6. Function, length 8 bits
7. Function instance, length 5 bits
8. ECU instance, length 3 bits
9. Manufacturer code, length 11 bits
10. Identity number, length 21 bits

Byte number in CAN message	Contents/Meaning
0	Identity number, LSB
1	Identity number
2	Bits 0-4: Identity number, MSB

	Bits 5-7: Manufacturer code, LSB
3	Manufacturer code, MSB
4	Bits 0-2: ECU instance Bits 3-7: Function instance
5	Function
6	Bit 0: Reserved bit Bits 1-7: Vehicle system
7	Bits 0-3: Vehicle system instance Bits 4-6: Industry group Bit 7: Arbitrary address bit

The main purpose of the Name is to describe an ECU. The lower Function field values, 0 to 127, are pre-assigned to "standard" functions or devices. The values 128 to 254 are dependent on the Industry Group and the Vehicle System values. This dependence makes it possible to have the same arrangement of functions in different vehicles. This system also allows devices such as trailers and agricultural equipment to limit their search for an available address and thus minimize the time and difficulty of dynamically claiming an address. When claiming an address, the Name is used to determine which ECU has higher priority and therefore will get the address that was claimed.

Each device on the network will be associated with at least one Name and one address. However, multiple device Names and multiple addresses may coexist within a single ECU. For example, an engine and engine brake (retarder) residing in a common device with a single physical bus connection. The device address defines a specific communications source or destination for messages. The Name identifies the functionality and adds a unique instance number of that functionality when multiple devices of the same type coexist on the network. Only 254 different devices of the same type can coexist on the network due the address limit. Address 255 is reserved as a global address for broadcast and address 254 is reserved as the "null address" used by devices that have not yet claimed an address or failed to claim an address.

Receiving Messages (J1939/21 and J1939/7x)

There are various techniques (and chips) available for capturing selected messages off the network. Several general observations can be made, however regarding received messages:

1. If a message is a destination specific request or command, the device must determine if the destination address matches an address claimed by the device. If there is a match, the receiving device must process the message and provide some type of acknowledgment.
2. If a message is a global request, every device, even the originator, must process the request and respond if the data is available.
3. If a message is broadcast, each device must determine if the content is of relevance or not.

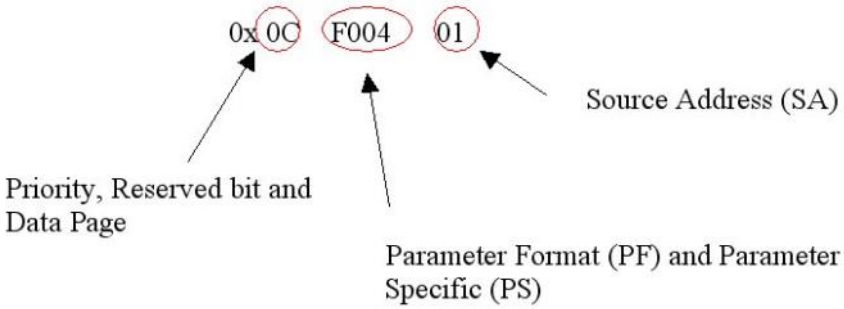
Example of how to interpret a J1939 message

This example is intended to provide the principles of how to interpret a J1939 message.
Let's look at a J1939 message with the following content:

CAN identifier: 0xCF00401

Data Bytes: 0xFF FF 82 DF 1A FF FF FF

What information does the CAN-ID provide?



First two bytes = 0x0C = 00001100 in binary format. The first 3 bits aren't used since the identifier only consists of 29 bits. The following 3 bits represents the message priority which in this case is 3. Thereafter follows a reserved bit and then the data page which are used to determine the complete PGN.

The last byte of the CAN-ID is the Source Address (address of the sending device) which in this case is 1.

The PGN = 0x0F004 which corresponds to the Electric Engine Controller #1 (EEC1) according to the J1939/71 document. This document also describes the parameters and their position within the data bytes. The data field consists of the following bytes in this example:

Data bytes	**FF**	**FF**	**82**	**DF**	**1A**	**FF**	**FF**	**FF**
Position	1	2	3	4	5	6	7	8

Data bytes 1, 2, 6, 7 and 8 in this example are not available and are therefore set to 0xFF. No raw parameter value (single byte) could have the value 0xFF.

Data byte 3 is the parameter Actual engine percent torque. The raw value 0x82 is 130 decimal. According to the J1939/71 document the scaling 1% per bit and the offset is -125. Therefore, the actual value for this parameter is 5%.

Data bytes 4 and 5 form the parameter Engine speed. The first byte (4) is the least significant (Intel byte order). The raw value 0x1ADF = 6879 decimal. The scaling is 0.125 rpm per bit and the offset is 0. The actual value for this parameter is therefore just under 859.875 rpm.

--
end