Open SAE J1939

Foreword

SAE J1939 is a protocol for shaping the CAN-bus message in a specific way that suits industrial vehicles such as tractors, machinery, trucks and more.

SAE J1939 is a very easy protocol to use, but there is a lack of information about SAE J1939, due to the cost of the protocol document, available how to shape a CAN-bus message according to SAE J1939 protocol standard. So therefore I'm writing a SAE J1939 protocol available for free to use on any embedded systems such as STM32, Arduino, AVR, PIC etc.

To learn to build on this project, you need first to understand SAE J1939. I have written this project in C language because C is an industry standard. The C language dialect I have chosen is C99 and I don't use dynamical memory allocation in this library. So it will work with MISRA C standard.

With this library, you can communicate with valves, engines, actuators, hardware and all other things that are suitable for heavy industrial mobile applications. I have build up a basic structure of the project and I hope that other users will send pull request of their C code for extra functionality to SAE J1939 standard because SAE J1939 is a huge standard.

- Daniel Mårtensson, Sweden, 2021-07-14

Quick introduction to SAE J1939 ID and DATA

First you need to understand the data frame of SAE J1939. The data frame of SAE J1939 is basically one ID and one DATA. ID and DATA is the information we are going to send through the CAN-bus network.

Figure 1: SAE J1939 Data frame

ID contains 4 hex values and DATA contains maximum 8 hex values e.g 8 bytes values.
An ID can be shaped as this 0x18FE8022 where 80 is the destination address, called DA and 22 is the source address, called SA. So you more likely going to see ID numbers as this 0x18FE'DA''SA'

The DA address and SA address is the addresses of the ECU – Electronic Control Unit, e.g microcontroller/PCB board. You self decide what address your ECU should have. For example. If you have two ECU that are going to communicate to each other. Then you have for example ECU1 address as 0xA1 and ECU2 as 0xA5. If ECU1 what to send a message to ECU2, then the destination address DA would be 0xA5 and the source address SA would be 0xA1. If ECU2 want to send a message to ECU1, then DA would be 0xA1 and source address SA would be 0xA5.

So keep that in mind that when I'm talking about SA and DA, it's only about source address and destination address of the ECU. When it comes to the rest if the ID values 0x18FE, they are just

there to describe the type of message. Each message have a unique ID and therefore the ID will be shaped differently. Sometimes, DA in the ID message is replaced by a unique hex value.

DATA is just 8 bytes of data. DATA holds information about anything the user want to send to other ECU. Sometimes DATA can be 3 bytes depending on if we are sending a unique message.

Functionality overview of Open SAE J1939

Open SAE J1939 is just basic C-code, arrays, function, structures and bitwise operations. Nothing more. I have shaped this project according to this map below. This map describe basic overview of the project. Some functions may have been added after this document was made, but the structure is the same. Those functions are inside the green box and they are called extra functionality of the project.

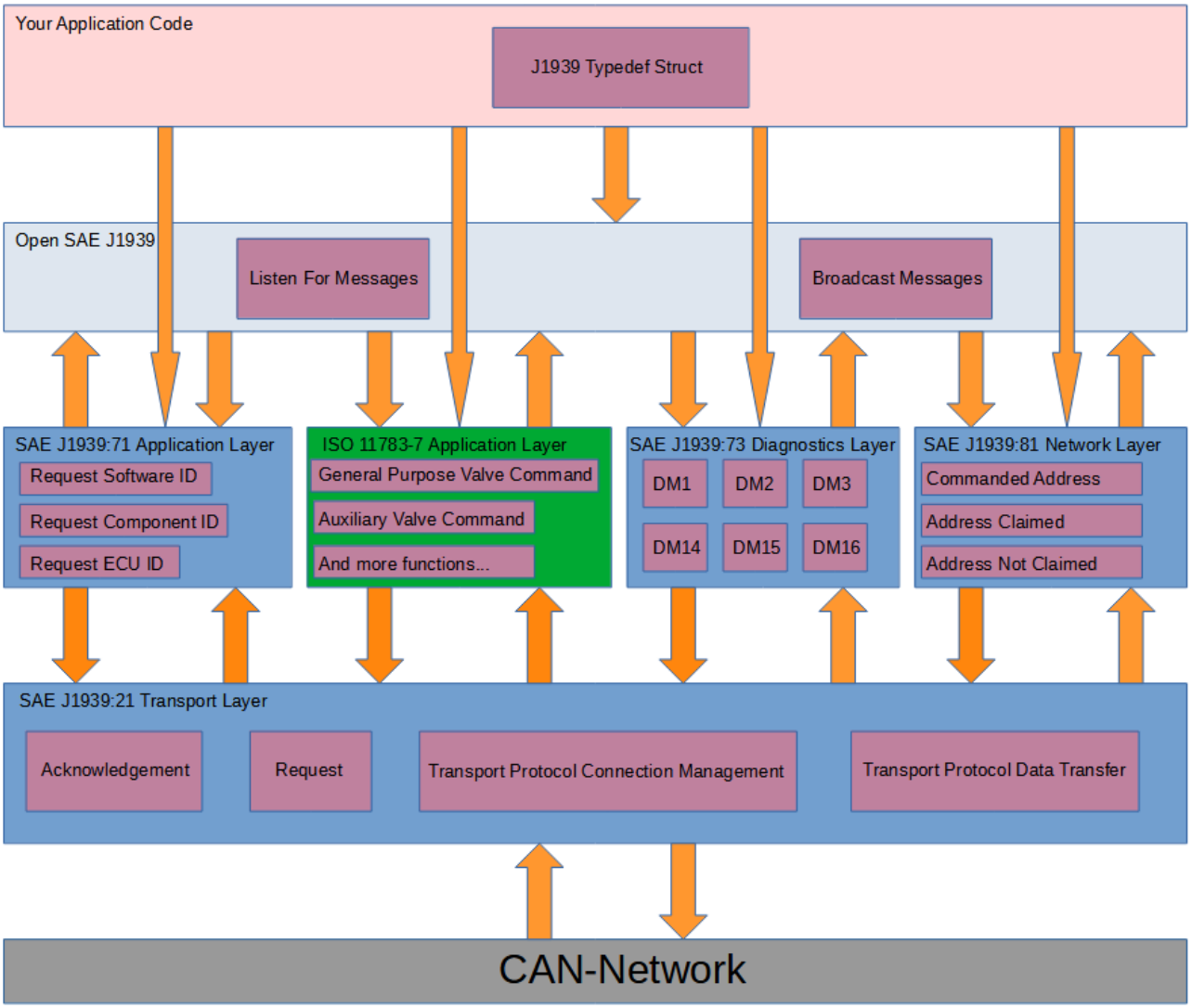

Figure 2: Structure of this project

We begin from the top. The J1939 Typedef Struct. That structure is just a basic C code typedef struct that holds information about all ECU who are connected to the CAN-network. The struct can be found at the Open SAE J1939 folder.

The idea behind the struct:

- The struct holds all information about all ECU.
- The user are going to create a C pointer of the struct and pass it to the SAE J1939 functions.
- For every ID DATA message the user get, the ID contains source address SA and SA is interpreted as an index number for each ECU for storing the DATA inside the struct at a specific index. So if SA is 0x3A = 58, then the struct placing all DATA information about that ECU at index 58 because that ECU has the address 58. Therefore the struct can hold multiple information about all ECU.

Open SAE J1939

Listen For Messages

This function is called all the time by the user. Most likely inside a while loop. This function reading ID and DATA directly and split ID into 4 different ID for determine what kind of ID it is and what to do next. It split up ID, check what ID it is and then check what function it want to call and then pass that DATA + source address SA + pointer of the struct to that function as argument. That's it. Listen For Messages is just a function who select which function we want to call, depending on how the ID looks like.

Broadcast Messages

According to the J1939 standard, some functions should be cyclic and called for example every 1000ms. These function broadcast messages to all ECU. It can be for example error status functions or measurement functions who broadcast their information to all other ECU for every X milliseconds. Here, you as the user, have right to add which function you want to be called at a specific time. This function must be called every fixed time step of milliseconds for example every 10ms or every 100ms.

SAE J1939:71 Application Layer

Request Software ID

Every ECU has a software ID for example a number or a text that describe for example BIOS software or software version. Call this function if you want to request the software ID from other ECU. The ECU will then response back to you with information.

Request Component ID

Every ECU has a component it control. It can be for example a motor, valve or other thing. Call this function if you want the ECU to send information about the component that ECU controls.

Request ECU ID

Same as above. Call this function if you want to know the ID information about the ECU you are sending to.

ISO 11783-7 Application Layer

Here I have selected the application layer of agriculture, tractors and machinery ISO standard. That application layer contains these functions:

- General Purpose Valve command – Sending control messages to ECU for controlling valves
- Auxiliary Valve Command – Sending control messages to ECU for controlling valves
- General Purpose Valve Estimated Flow – Get the computed flow from a valve
- Auxiliary Valve Estimated Flow – Get the computed flow from a valve
- Auxiliary Valve Estimated Position – Get the computed position of the valve main spool

SAE J1939:73 Diagnostics Layer

Diagnostics layer contains messages about errors, but it also can contains functionality such as memory access of the EEPROM (built in “hard drive”) or command for deleting error messages.

- DM1 – Contains error messages such as electrical fault, location etc
- DM2 – Contains previous active DM1 messages
- DM3 – Deleting DM2 messages
- DM14 – Memory request of bytes, address etc from a ECU
- DM15 – Memory response from a DM14 request from that ECU
- DM16 – Binary data transfer (if DM15 repose was OK)

SAE J1939:81 Network Layer

At the startup of an ECU, the ECU sending Address Claimed, which means it giving out the information about something called NAME. NAME is basic information about the ECU + SA.

- Address Claimed – Send out NAME + ID that holds SA.
- Address Not Claimed – Send out NAME + ID that holds Address Not Claimed.
- Commanded Address – Send this to an ECU and you can change its NAME and SA.

SAE J1939:21 Transport Layer

- Acknowledgement – Send a OK, BUSY, WAIT etc. as a response on a request.
- Request – Asking information or functionality about an ECU
- Transport Protocol Connection Management – Send how much data in packages you want to send to the receiver.
- Transport Protocol Data Transfer – Send the data in packages to the receiver.

SAE J1939 Protocol - Basics

Request

Request are used when we want to ask an ECU of “something”. That “something” is a PGN number. Enter a PGN number, send the message and get a response from that ECU directly. All PGN numbers can be found in SAE_J1939_Enum_PGN.h

Message ID	18 EA "DA" "SA"
PGN	0x00EA00 (59904)
Peer to Peer/Broadcast	Peer to Peer (could become broadcast if DA = 255)
Message length	3 bytes
Multipacket	No
Byte 1	PGN LSB
Byte 2	PGN
Byte 3	PGN MSB

In case you did not understand what LSB and MSB is. LSB stands for least signifiant bit and MSB stands for most signifiant bit. LSB means the lowest number and MSB means the highest number. If we want to send the PGN number 0x18EEA1, then the LSB is 0xA1 because that’s the lowest number and MSB is 0x18 because that’s the highest number. The same reason when you write out 1024 where you begin with highest number 1(1000), then 0(000), then 2(20) and last 4. 1000 + 000 + 20 + 4 = 1024. So keep that in mind that MSB is the highest number and LSB is the lowest number.

Acknowledgement

Acknowledgement are used when we want to response back to an ECU e.g completed, busy, not available etc.

Message ID	18 E8 "DA" "SA"
PGN	0x00E800 (59392)
Peer to Peer/Broadcast	Peer to Peer (could become broadcast if DA = 255)
Message length	8 bytes
Multipacket	No
Byte 1	Control byte
Byte 2	Group function value
Byte 3	0xFF (Reserved)
Byte 4	0xFF (Reserved)

Byte 5	Address
Byte 6	PGN of requested info LSB
Byte 7	PGN of requested info
Byte 8	PGN of requested info MSB

- Control byte: Describes the status of the requested PGN. Have a look at SAE_J1939_Enum_Control_Byte.h
- Group function value: The function code that specify the cause of the control byte. Have a look at SAE_J1939_Enum_Group_Function_Value.h
- Address: The source address of the ECU
- PGN of requested info: The same PGN number in the request. Have a look at SAE_J1939_Enum_PGN.h

Transport Protocol Connection Management

This describe for the receiver ECU how many packages and total size of data we are going to send.

Message ID	1C EC "DA" "SA"
PGN	0x00EC00 (60416)
Peer to Peer/Broadcast	Peer to Peer (could become broadcast if DA = 255)
Message length	8 bytes
Multipacket	No
Byte 1	Control byte
Byte 2	Total message size LSB
Byte 3	Total message size MSB
Byte 4	Number of packages
Byte 5	0xFF (Reserved)
Byte 6	PGN of the packeted message LSB
Byte 7	PGN of the packeted message
Byte 8	PGN of the packeted message MSB

- Control byte: How the transport protocol data transfer should work. Have a look at SAE_J1939_Enum_Control_Byte.h
- Total message size: How many bytes we want to send
- Number of packages: In how many packages we are going to send. Rule of thumb is if total_message_size % 8 > 1, then number_of_packages = total_message_size/8 + 1, else number_of_packages = total_message_size/8 (% means modulus)

- PGN of the packeted message: What type of function do we want the ECU receiver to do.

Here is two way to send a multi pack messages:

- BAM – Just send the message to the ECU
- RTS/CTS – Wait for a response (With an acknowledgement response at the end)

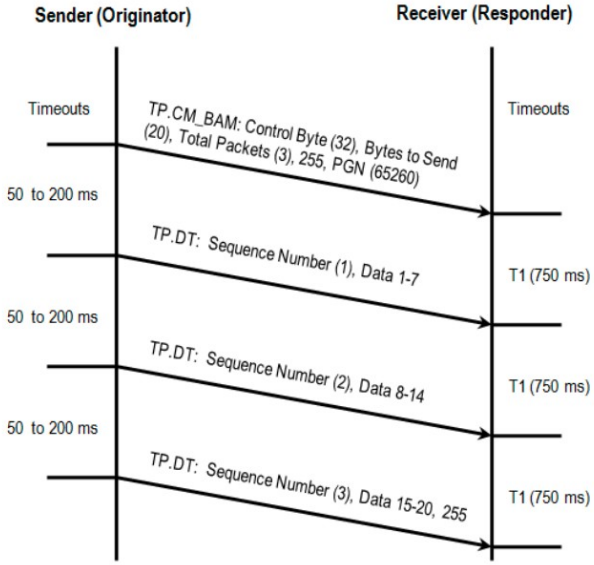

Figure 3: Broadcast Announce Message(BAM)

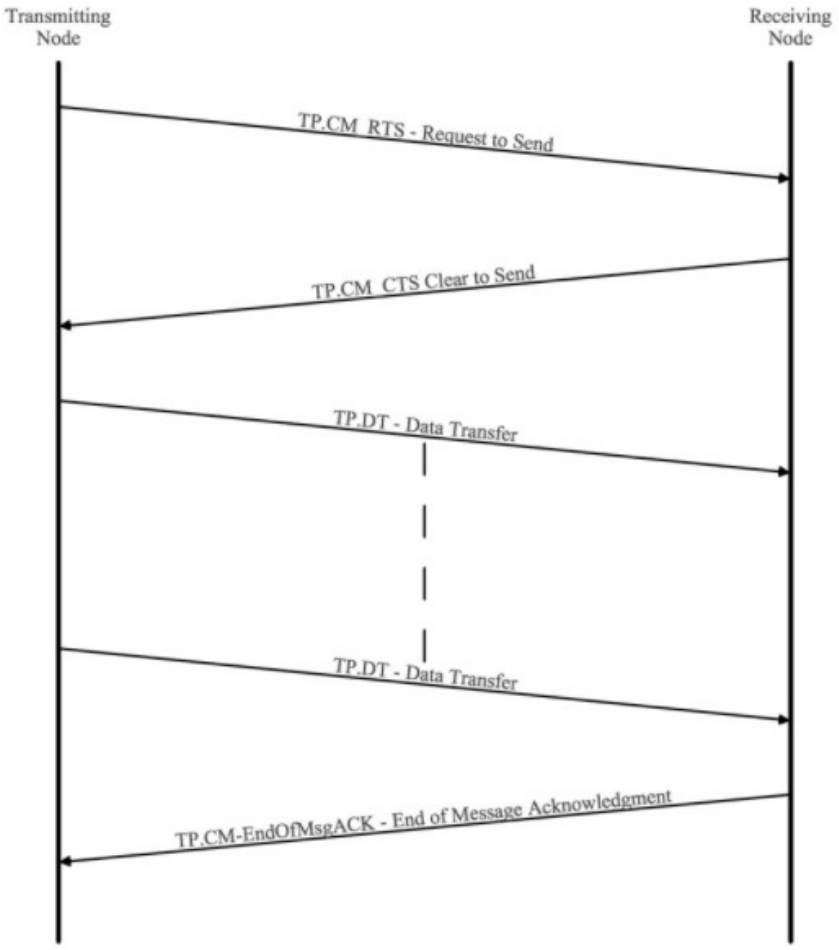

Figure 4: Request To Send(RTS), Clear To Send(CTS) and EndOfMsgACK

Transport Protocol Data Transfer

This sending packages to the receiver ECU. Transport Protocol Connection Management must be sent first. Else the receiver won’t know how many packages and total data and PGN number we the transmitter is sending.

Message ID	1C EB "DA" "SA"
PGN	0x00EB00 (60160)
Peer to Peer/Broadcast	Peer to Peer (could become broadcast if DA = 255)
Message length	8 bytes
Multipacket	No
Byte 1	Sequence number
Byte 2	Data 1
Byte 3	Data 2
Byte 4	Data 3
Byte 5	Data 4
Byte 6	Data 5
Byte 7	Data 6
Byte 8	Data 7

- Sequence number: This is the order/index of the data transfer packages
- Data 1 to Data 7: The data you want to transfer. If you only have 10 bytes to send. It will be 2 packages in total, but the rest of the unused bytes will be 0xFF.

Address Claimed

This describes the NAME about the ECU, and also the other ECU get to know its SA.

Message ID	18 EE "DA" "SA"
PGN	0x00EE00 (60928)
Peer to Peer/Broadcast	Peer to Peer (could become broadcast if DA = 255)
Message length	8 bytes
Multipacket	No
Repetition rate	Sent at power on, after a “Request” for “Address Claim” and after a “Commanded Address”
Byte 1	Identity number LSB
Byte 2	Identity number
Byte 3	Manufacturer code LSB [8..6], Identity number MSB [5..1]

Byte 4	Manufacturer code MSB
Byte 5	Function instance [8..4], ECU Instance [3..1]
Byte 6	Function
Byte 7	Vehicle system [8..2], 1 Reserved [1]
Byte 8	Arbitrary address capable [8], Industry group [7..5], Vehicle system instance [4..1]

Notice that [X..Y] describes the bit index in the byte. It more likely looks like this.

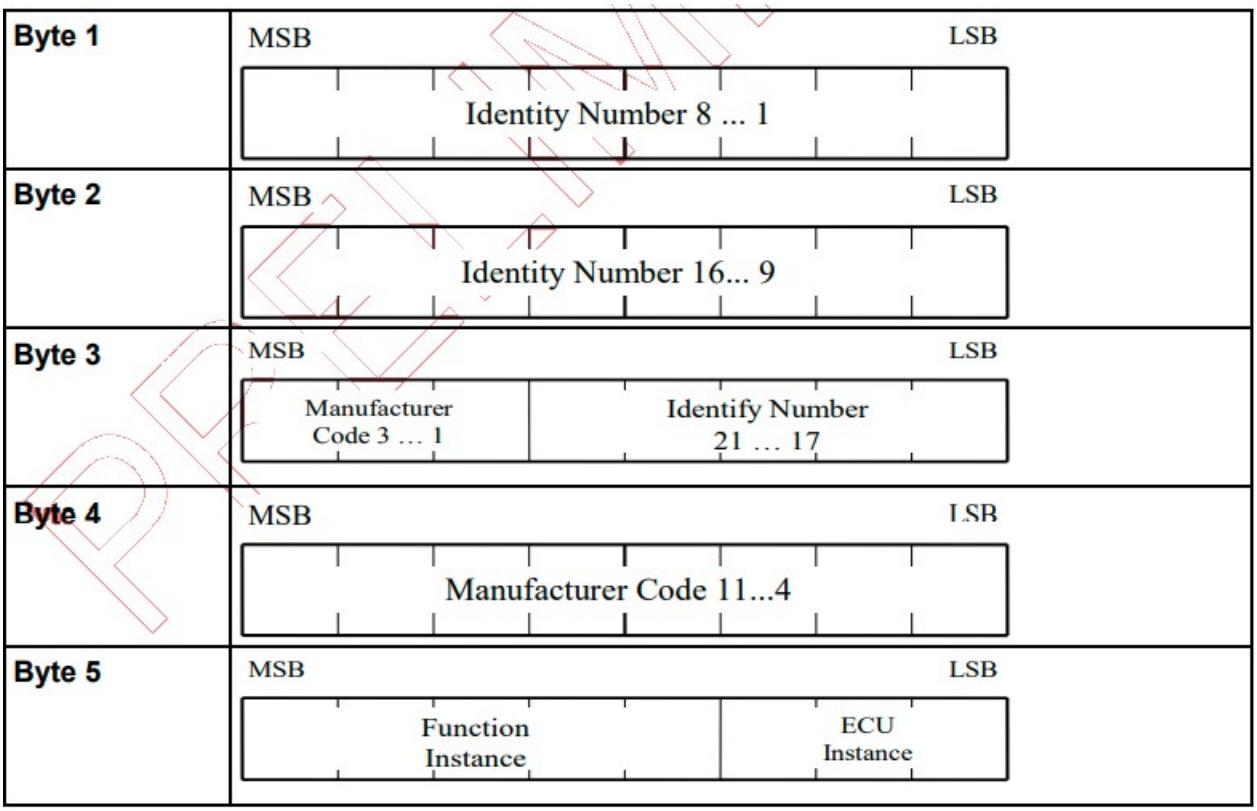

Figure 5: How to interpret

- Identity number: Specify the ECU serial ID – 0 to 2097151
- Manufacturer code: Specify the ECU manufacturer code – 0 to 2047
- Function instance: Specify the ECU function number – 0 to 31
- ECU instance: Specify the ECU number – 0 to 7
- ECU function: Specify the ECU function – 0 to 255
- Vehicle system: Specify the type of vehicle where ECU is located – 0 to 127
- Arbitrary address capable: Specify if the ECU have right to change address if addresses conflicts – 0 to 1
- Industry group: Specify the group where this ECU is located – 0 to 7
- Vehicle system instance: Specify the vehicle system number – 0 to 15

Have a look at SAE_J1939_Enum_NAME.h

Address not claimed

When an ECU cannot claim their address, the other ECU will receive this message. The user don’t know which ECU cannot claim their address. The user can only see the amount of how many ECU could not claim their address. See the uint8_t variable number_of_cannot_claim_address inside the struct J1939.

Message ID	18 EE "DA" "SA"
PGN	0x00EE00 (60928)
Peer to Peer/Broadcast	Peer to Peer (could become broadcast if DA = 255)
Message length	8 bytes
Multipacket	No
Repetition rate	• Sent at ECU start up • On request from other ECU • After Commanded Address
Byte 1	Identity number LSB
Byte 2	Identity number
Byte 3	Manufacturer code LSB [8..6], Identity number MSB [5..1]
Byte 4	Manufacturer code MSB
Byte 5	Function instance [8..4], ECU Instance [3..1]
Byte 6	Function
Byte 7	Vehicle system [8..2], 1 Reserved [1]
Byte 8	Arbitrary address capable [8], Industry group [7..5], Vehicle system instance [4..1]

Commanded Address

This command is used when the user want to change the NAME + SA at an ECU. Notice that this PGN don’t have an ID because Commanded Address DATA is 9 bytes.

PGN	0x00FED8 (65240)
Peer to Peer/Broadcast	Peer to Peer (could become broadcast if DA = 255)
Message length	8 bytes
Multipacket	Yes. Use Transport Protocol Connection Management → Transport Protocol Data Transfer
Byte 1	Identity number LSB
Byte 2	Identity number
Byte 3	Manufacturer code LSB [8..6], Identity number MSB [5..1]
Byte 4	Manufacturer code MSB

Byte 5	Function instance [8..4], ECU Instance [3..1]
Byte 6	Function
Byte 7	Vehicle system [8..2], 1 Reserved [1]
Byte 8	Arbitrary address capable [8], Industry group [7..5], Vehicle system instance [4..1]
Byte 9	New ECU address

- New ECU address: Specify the new ECU address – 0 to 254 because 255 = Broadcast

DM1

Diagnostics of error and location of an ECU. Notice that this message ID don’t have a destination address because it’s a broadcast message to all ECU.

Message ID	18 FE CA "SA"
PGN	0x00FECA (65226)
Peer to Peer/Broadcast	Broadcast
Message length	8 bytes
Multipacket	No: Normal transfer Yes: Use Transport Protocol Connection Management → Transport Protocol Data Transfer
Repetition rate	• Broadcast each second • On request from another ECU
Byte 1	SAE Lamp status malfunction indicator [8..7], SAE Lamp status red stop [6..5], SAE Lamp status amber warning [4..3], SAE lamp status protect lamp [2..1]
Byte 2	SAE Flash lamp malfunction indicator [8..7], SAE Flash lamp red stop [6..5], SAE Flash lamp amber warning [4..3], SAE Flash lamp protect lamp [2..1]
Byte 3	SPN LSB
Byte 4	SPN
Byte 5	SPN MSB [8..6], FMI [5..1]
Byte 6	SPN conversion method [8], Occurence count[7..1]
Byte 7	0xFF (Reserved)
Byte 8	0xFF (Reserved)

- SAE Lamp status malfunction indicator: SAE lamp status indicates fault – 0 to 1
- SAE Lamp status red stop: SAE lamp light up red – 0 to 1
- SAE Lamp status amber warning: SAE lamp lights up with amber warning – 0 to 1
- SAE lamp status protect lamp: SAE light up protect lamp – 0 to 1

The analogy is the same for SAE Flash lamp, it's just a different SAE lamp.

- SPN = Suspect Parameter Number e.g the location of the fault. Have a look at SAE_J1939_Enum_DM1.h
- FMI = Failure Mode Identifier e.g what causing the error code. Have a look at SAE_J1939_Enum_DM1.h
- SPN Conversion method: 1 = Diagnostics Trouble Code are aligned using a newer conversion method. 0 = One of the three Diagnostics Trouble Code conversion methods is used and ECU manufacture shall know which of the three methods is used
- Occurence count: This is how often the DM1 error message becomes active – 0 to 126

DM2

DM2 is the previous active DM1 messages.

Message ID	18 FE CB "SA"
PGN	0x00FECB (65227)
Peer to Peer/Broadcast	Broadcast
Message length	8 bytes
Multipacket	No: Normal transfer Yes: Use Transport Protocol Connection Management → Transport Protocol Data Transfer
Repetition rate	On request from other ECU
Byte 1	SAE Lamp status malfunction indicator [8..7], SAE Lamp status red stop [6..5], SAE Lamp status amber warning [4..3], SAE lamp status protect lamp [2..1]
Byte 2	SAE Flash lamp malfunction indicator [8..7], SAE Flash lamp red stop [6..5], SAE Flash lamp amber warning [4..3], SAE Flash lamp protect lamp [2..1]
Byte 3	SPN LSB
Byte 4	SPN
Byte 5	SPN MSB [8..6], FMI [5..1]
Byte 6	SPN conversion method [8], Occurence count[7..1]
Byte 7	0xFF (Reserved)
Byte 8	0xFF (Reserved)

DM3

DM3 is a request to clear DM2 messages. Use the request function above. After this is done, DM2 will be sent to all ECU via broadcast.

PGN	0x00FECC (65228)

DM11 – Not supported

DM11 is a request for clearing DM1 messages. The reason why I don’t have implement it, is because in my opinion, it’s just an administrative burden that other ECU need to clear the DM1 error codes. I want the ECU it self to clear their own DM1 codes. The same way it activate the error codes by it self.

Software identification

Send out the software ID of the ECU. Notice that DA here in the message ID is not destination address. It’s the hex value 0xDA.

Message ID	18 FE DA "SA"
PGN	0x00FEDA (65242)
Peer to Peer/Broadcast	Broadcast
Message length	8 bytes
Multipacket	No: Normal transfer Yes: Use Transport Protocol Connection Management → Transport Protocol Data Transfer
Repetition rate	On request from other ECU
Byte 1	Number of fields
Byte 2	Identification 1
Byte 3	Identification 2
Byte 4	Identification 3
Byte 5	Identification 4
Byte 6	Identification 5
Byte 7	Identification 6
Byte 8	Identification 7

- Number of fields: How many identifications we have. If number of fields exceed 7, then we need to use multipacket transfer
- Identification 1-7: Most likely ASCII characters

ECU identification

Send out ECU identification to the ECU. Notice that there are a field called length_of_each_field in ECU_identification inside the J1939 struct. If length_of_each_field = 1, then this message is going to be transfer in a normal way, else multipacket.

Message ID	18 FD C5 "SA"
PGN	0x00FDC5 (64965)
Peer to Peer/Broadcast	Broadcast
Message length	8 bytes
Multipacket	No: Normal transfer Yes: Use Transport Protocol Connection Management → Transport Protocol Data Transfer
Repetition rate	On request from other ECU
Byte 1	ECU part number
Byte 2	ECU serial number
Byte 3	ECU location
Byte 4	ECU type
Byte 5	ECU manufacturer
Byte 6	ECU hardware version
Byte 7	0xFF (Reserved)
Byte 8	0xFF (Reserved)

All these field above are ASCII strings for describing the identification for the ECU.

Component identification

Send out component identification to the ECU. Notice that there are a field called length_of_each_field in Component_identification inside the J1939 struct. If length_of_each_field = 1, then this message is going to be transfer in a normal way, else multipacket.

Message ID	18 FE EB "SA"
PGN	0x00FEEB (65259)
Peer to Peer/Broadcast	Broadcast
Message length	8 bytes
Multipacket	No: Normal transfer Yes: Use Transport Protocol Connection Management → Transport Protocol Data Transfer
Repetition rate	On request from other ECU
Byte 1	Component product date

Byte 2	Component model name
Byte 3	Component model number
Byte 4	Component unit name
Byte 5	0xFF (Reserved)
Byte 6	0xFF (Reserved)
Byte 7	0xFF (Reserved)
Byte 8	0xFF (Reserved)

All these field above are ASCII strings for describing the identification for the component.

DM14 & DM15 and DM16

DM14 is a memory request. DM15 is a memory response for the DM14 memory request and if DM15 response was OK, then DM16 data transfer is called next.

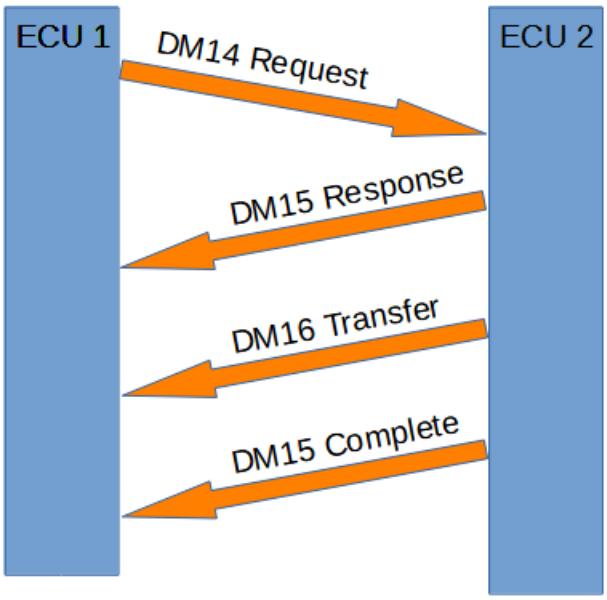

Figure 5: DM14, DM15 and DM16

DM14 memory request:

Message ID	18 D9 "DA" "SA"
PGN	0x00D900 (55552)
Peer to Peer/Broadcast	Peer to Peer (could become broadcast if DA = 255)
Message length	8 bytes
Multipacket	No
Byte 1	Number of requested bytes LSB
Byte 2	Number of requested bytes MSB [8..6], Pointer type [7], Command

	[6..3], 1 (Reserved) [1]
Byte 3	Pointer LSB
Byte 4	Pointer
Byte 5	Pointer MSB
Byte 6	Pointer extension
Byte 7	Key LSB
Byte 8	Key MSB

- Number of requested bytes: How many bytes we want to request from the ECU – 0 to 2047
- Pointer type: 0 if Pointer and Pointer extension are together one addresses. 1 if pointer is an address and Pointer extension is a way to describe a function code. Have a look at SAE_J1939_Enum_DM14_DM15.h
- Command: Write, read etc. Have a look at SAE_J1939_Enum_DM14_DM15.h
- Pointer: Flash, EEPROM etc address – 0 to 16777215
- Pointer extension: Extra pointer, or function code – 0 to 255
- Key: For example a number password or no password at all. Have a look at SAE_J1939_Enum_DM14_DM15.h

DM15 memory response of the DM14 memory request:

Message ID	18 D8 "DA" "SA"
PGN	0x00D800 (55296)
Peer to Peer/Broadcast	Peer to Peer (could become broadcast if DA = 255)
Message length	8 bytes
Multipacket	No
Repetition rate	On request from other ECU
Byte 1	Number of allowed bytes LSB
Byte 2	Number of allowed bytes MSB [8..6], 1(Reserved) [7], Status [6..3], 1 (Reserved) [1]
Byte 3	EDC parameter LSB
Byte 4	EDC parameter
Byte 5	EDC parameter MSB
Byte 6	EDCP extension
Byte 7	Seed LSB
Byte 8	Seed MSB

- Number of allowed bytes: The bytes the ECU are allowed to get. This is an answer on Number of requested bytes – 0 to 16777215
- Status: The status for the request e.g busy, operation failed, proceed, operation completed etc. Have a look at SAE_J1939_Enum_DM14_DM15.h
- EDC parameter: This is an explanation of the status response e.g Writing proceed now or Cannot write to a specific address etc. Have a look at SAE_J1939_Enum_DM14_DM15.h
- EDCP extension: If 0xFF, then Error indicator EDC parameter not used. Have a look at SAE_J1939_Enum_DM14_DM15.h
- Seed: Tells the ECU about the status of the key request

DM16 is binary data transfer:

Message ID	18 D7 "DA" "SA"
PGN	0x00D700 (55040)
Peer to Peer/Broadcast	Peer to Peer (could become broadcast if DA = 255)
Message length	8 bytes
Multipacket	No: Normal transfer Yes: Use Transport Protocol Connection Management → Transport Protocol Data Transfer
Repetition rate	After DM15 request was OK
Byte 1	Number of occurences
Byte 2	Raw binary data 1
Byte 3	Raw binary data 2
Byte 4	Raw binary data 3
Byte 5	Raw binary data 4
Byte 6	Raw binary data 5
Byte 7	Raw binary data 6
Byte 8	Raw binary data 7

- Number of occurences: How many raw binary data we have. If number of occurences exceed 7, then we need to use multipacket transfer
- Raw binary data 1-7: Binary data for memory implementation