

Intro to Truck Networks



**Hannah
Silva**



Senior Security
Consultant



Leviathan Security
Group

Agenda

Overview of Truck Systems

Connection Guide

Controller Area Network (CAN)

SAE J1939

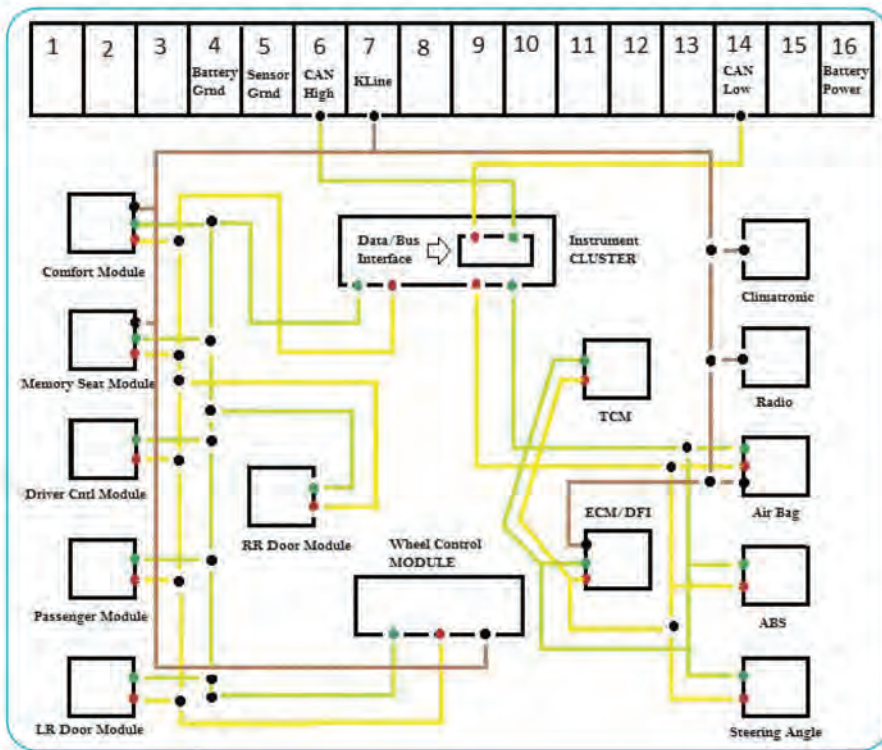
SAE J1708/J1587

SAE J2497

Why do we care?



What can we do?



Engine Control Module

Powertrain Control Module

Transmission Control Unit

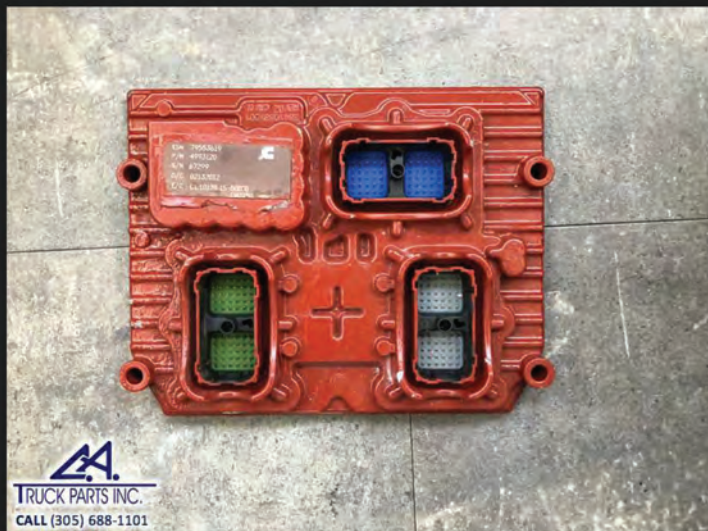
Brake Controller

Instrument Cluster

Body Controller

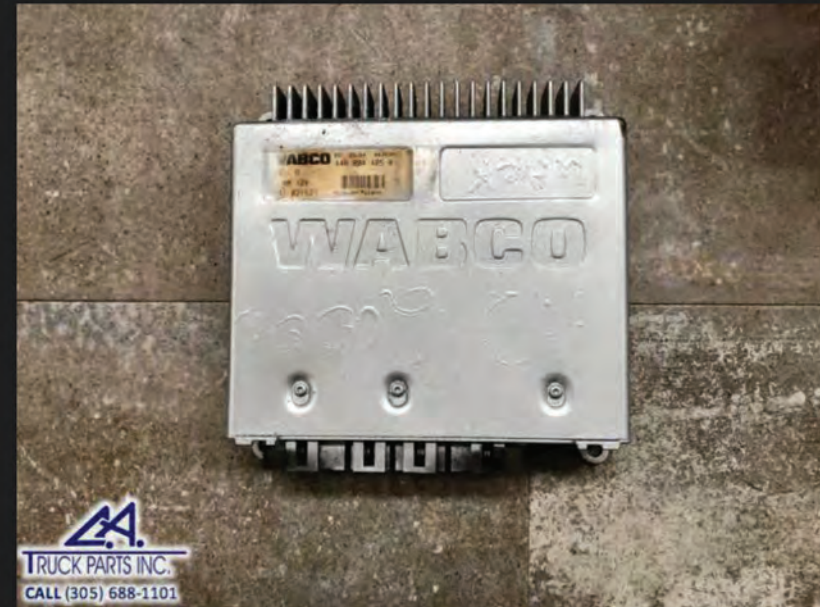
Truck Systems

ECM/PCM





Brake Controllers



Instrument Clusters


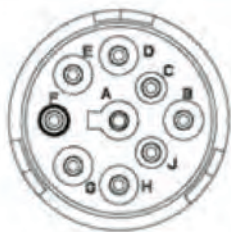


Getting Connected

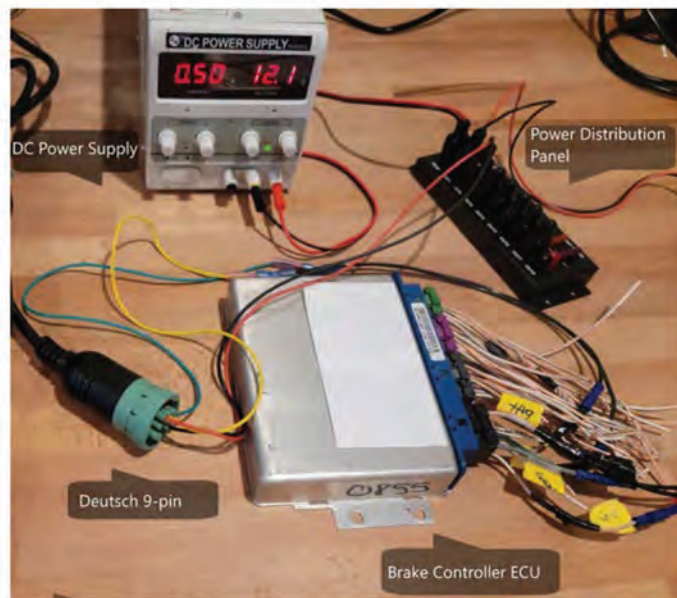


Physical Access to BUS

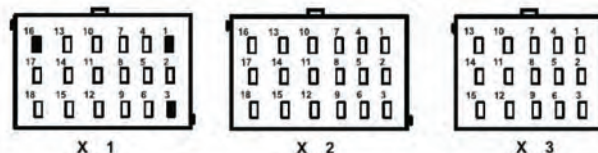
- Black is older, does not work with Green vehicle connector
- Green is backwards compatible with Black vehicle connector

		
Type 1 (Black)	Pin	Type 2 (Green)
Ground	A	Ground
Battery	B	Battery
J1939 + 250kb	C	J1939 + 500kb
J1939 - 250kb	D	J1939 - 500kb
J1939 Shield	E	J1939 Shield
J1708+	F	J1708+ / J1939 + 250kb
J1708-	G	J1708- / J1939 - 250kb
OEM Specific	H	OEM Specific
OEM Specific	J	OEM Specific

Deutsch 9-pin Connector

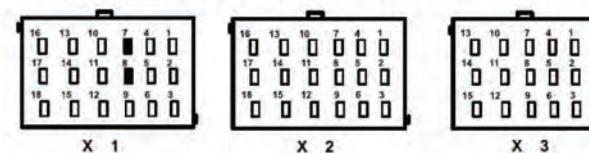


Cab-mount ECU: Looking into wire harness connector



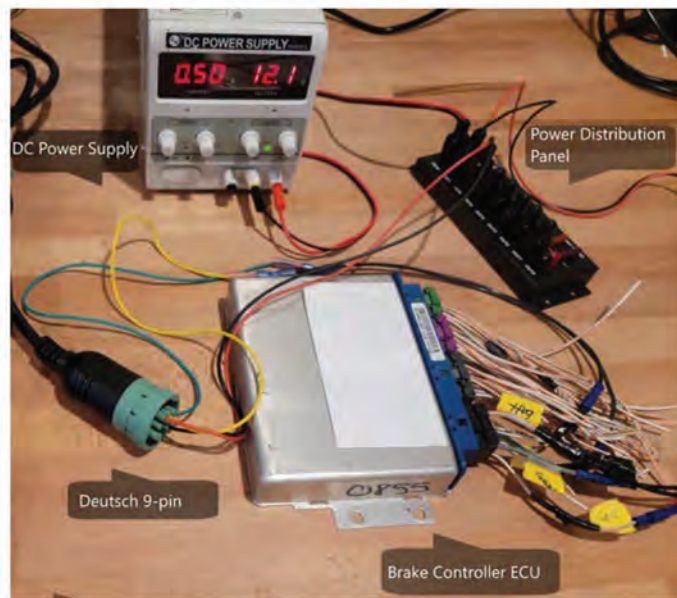
Connector	Pin	Power Supply Test
X1	1	Ground
18 Way	3	Ignition
	16	Battery

Cab-mount ECU: Looking into wire harness connector

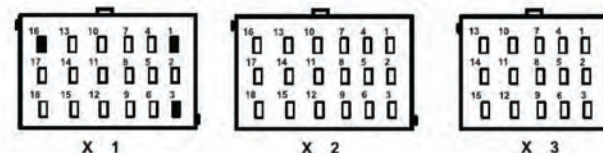


Connector	Pin	J1939
X1	7	J1939 Low
18 Way	8	J1939 High

Test Bench



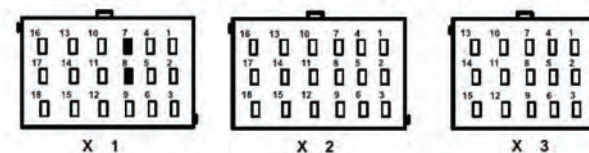
Cab-mount ECU:
Looking into wire harness connector



Connector	Pin	Power Supply Test
X1	1	Ground
18 Way	3	Ignition
	16	Battery

1 -> ground
3&16 -> +12V

Cab-mount ECU:
Looking into wire harness connector



Connector	Pin	J1939
X1	7	J1939 Low
18 Way	8	J1939 High

7 -> pin D (J1939 Low)
8 -> pin C (J1939 High)

Test Bench



Remote Access



WARNING: Technical Details Ahead

- Don't worry! You likely won't have to manually decode network packets
- Lots of tools out there to help
- These slides will be available!

Truck Networks

CAN (Controller Area Network)
- Physical and Data Link layers

J1939
- Higher layer protocol based on CAN
- Newer & faster

J1708
- Physical layer
- Primarily on older vehicles

J1587
- Higher layer protocol that runs on J1708

J2497
(PLC4TRUCKS)
- Basically J1708 but on trailer power lines

Truck Networks

CAN (Controller Area Network)
- Physical and Data Link layers

J1708

- Physical layer
- Primarily on older vehicles

J2497

(PLC4TRUCKS)
- Basically J1708 but on trailer power lines

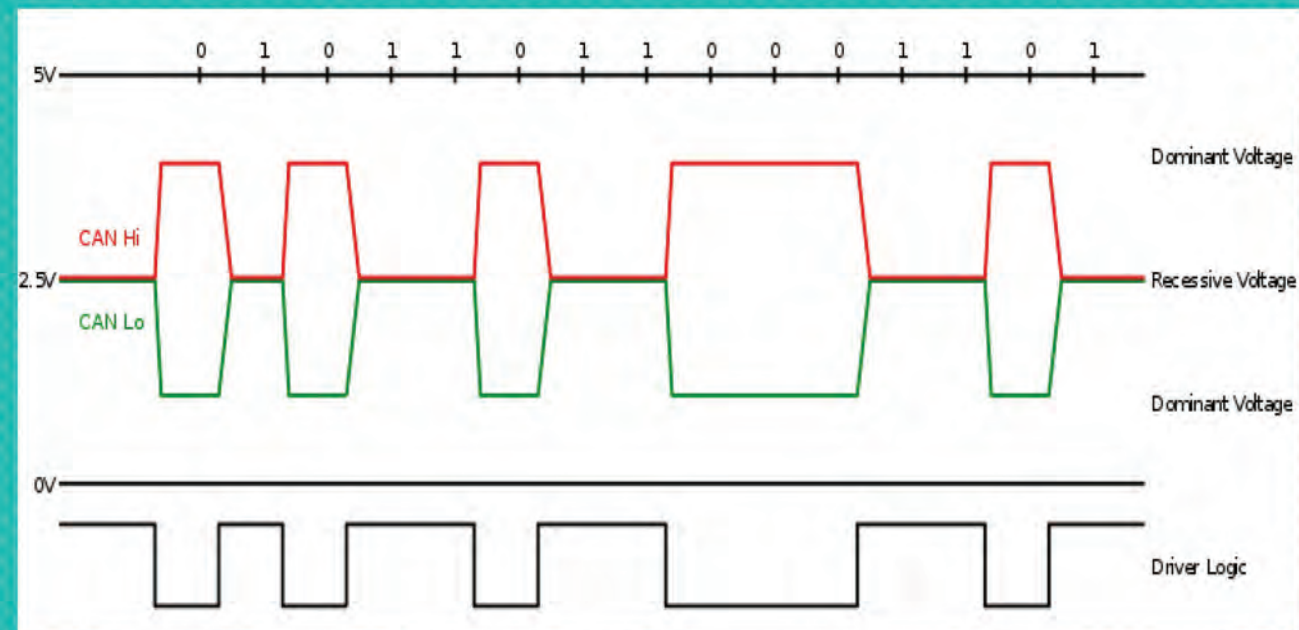
J1939

- Higher layer protocol based on CAN
- Newer & faster

J1587


- Higher layer protocol that runs on J1708

- Published in 1991
- Multi-master serial bus for ECU communication
- Nodes (ECUs) are physically connected by two twisted pair wires – CANH and CANL
- CANH and CANL stay at recessive state passively due to the 120 Ω terminating resistors placed at each end of the bus.
- Up to 1Mbit/s transmission speed



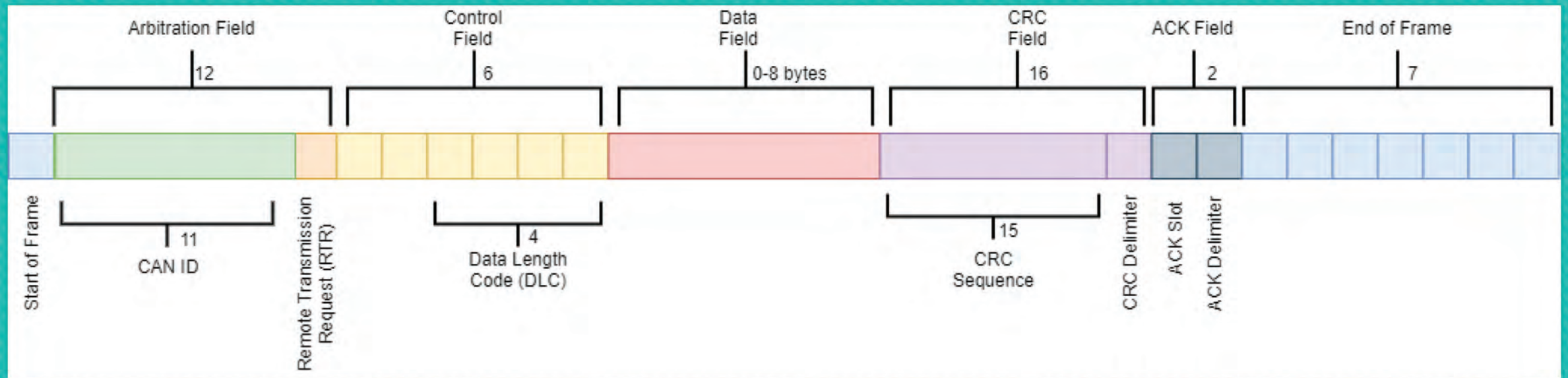
By EE JRW

CAN 2.0

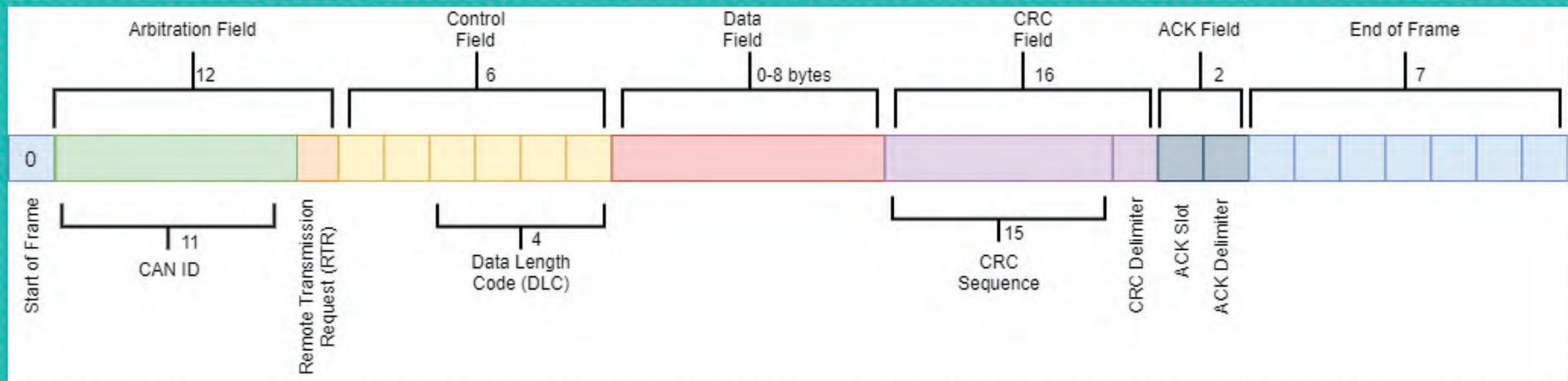
- 
- Data frame
 - Remote frame
 - Error frame
 - Overload frame

- CAN 2.0A: 11-bit CAN ID
- CAN 2.0B: 29-bit CAN ID

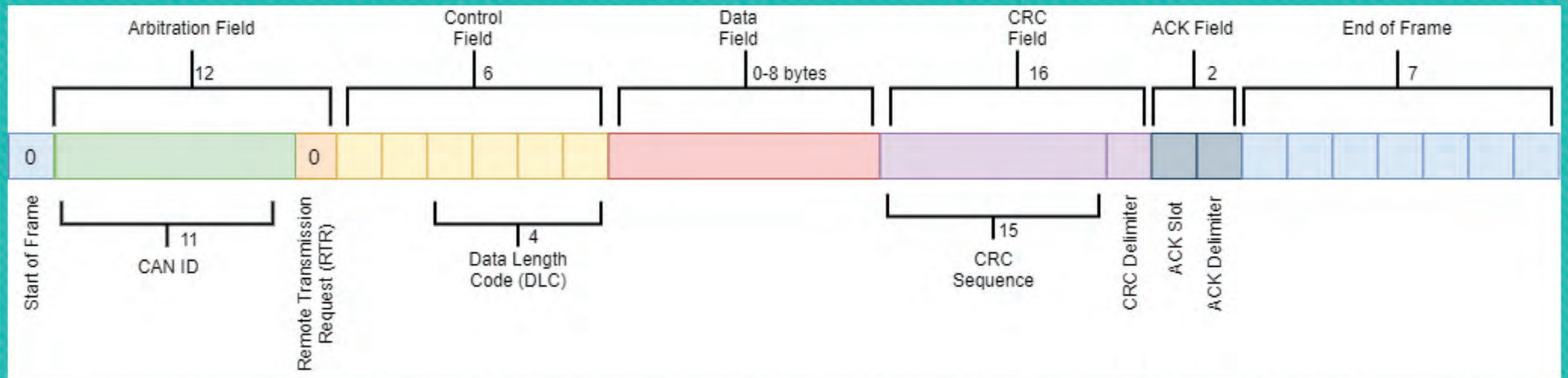
CAN Frames



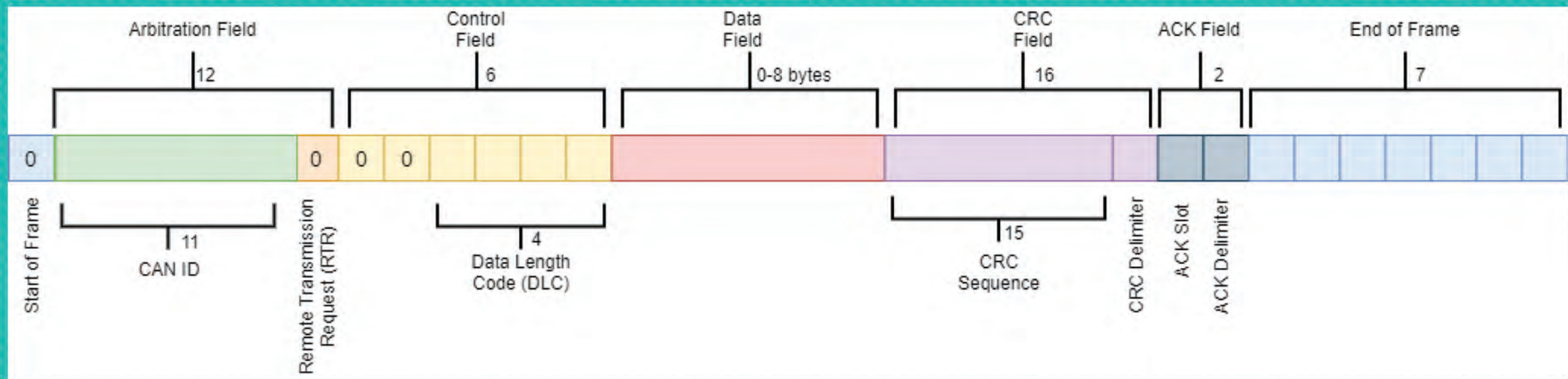
CAN Data Frame



CAN Data Frame - SOF

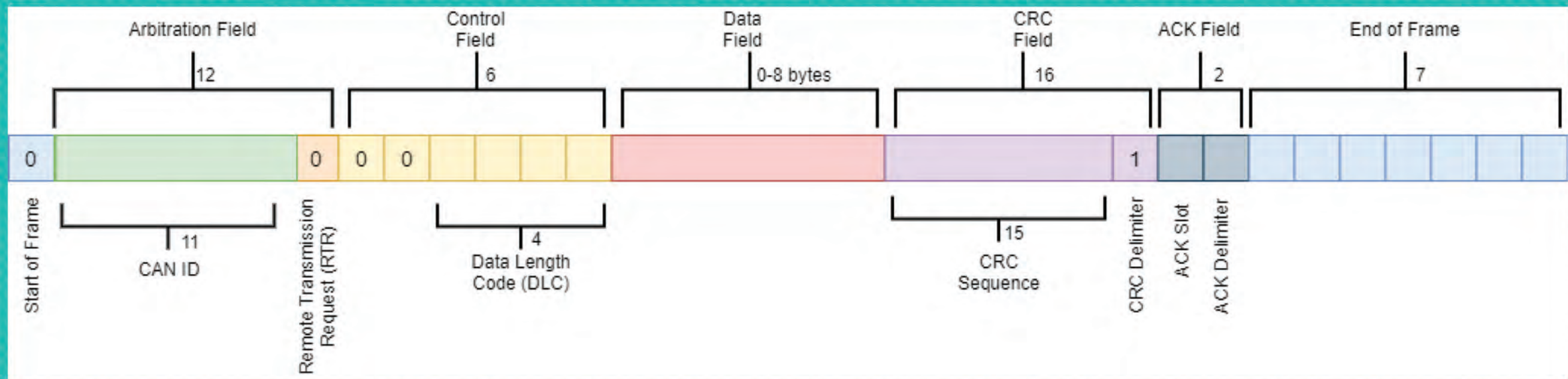


CAN Data Frame - RTR

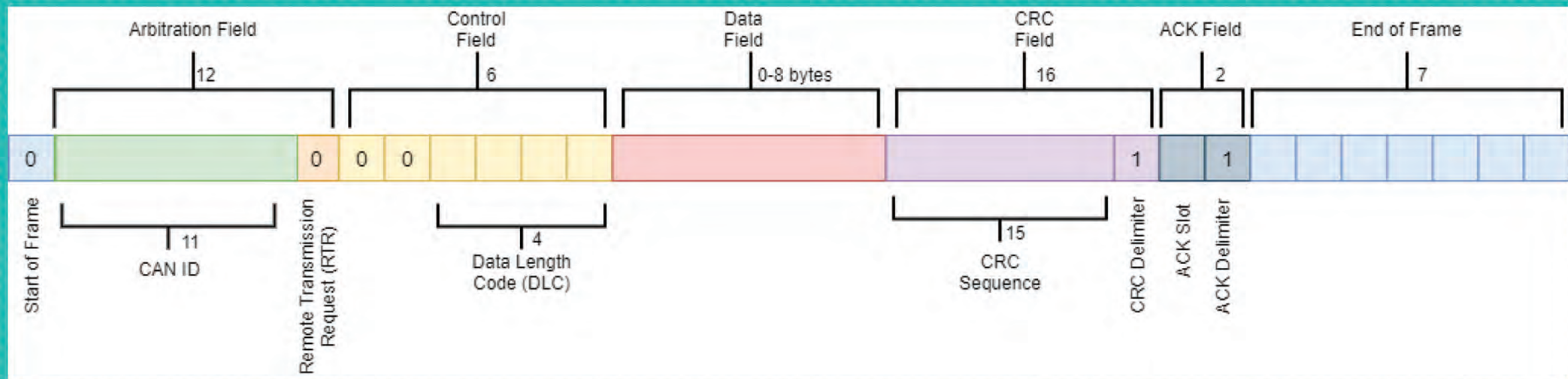


DLC: 0b0000 – 0b1000 (0-8)

CAN Data Frame – Control and Data Fields



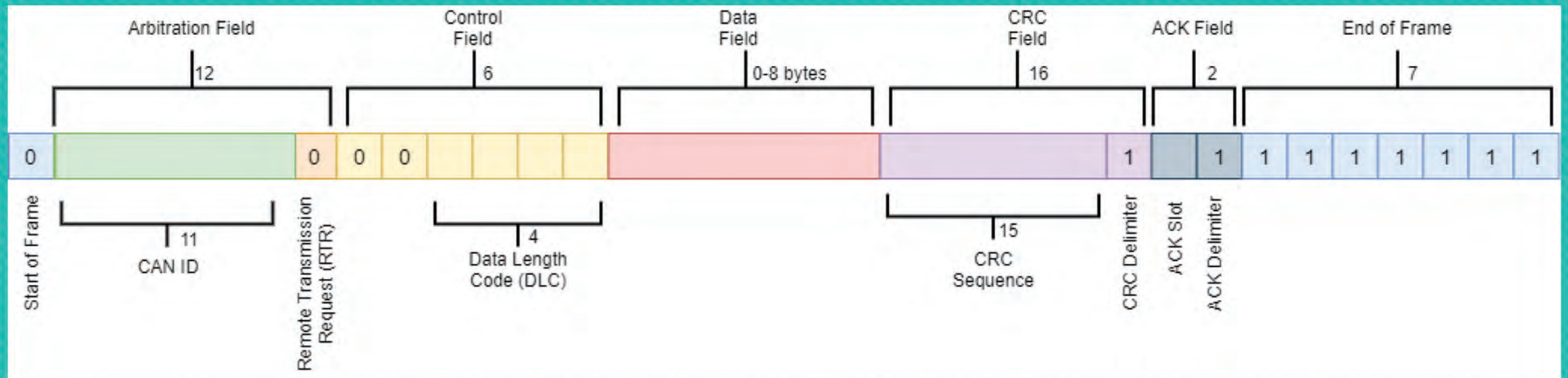
CAN Data Frame – CRC Field



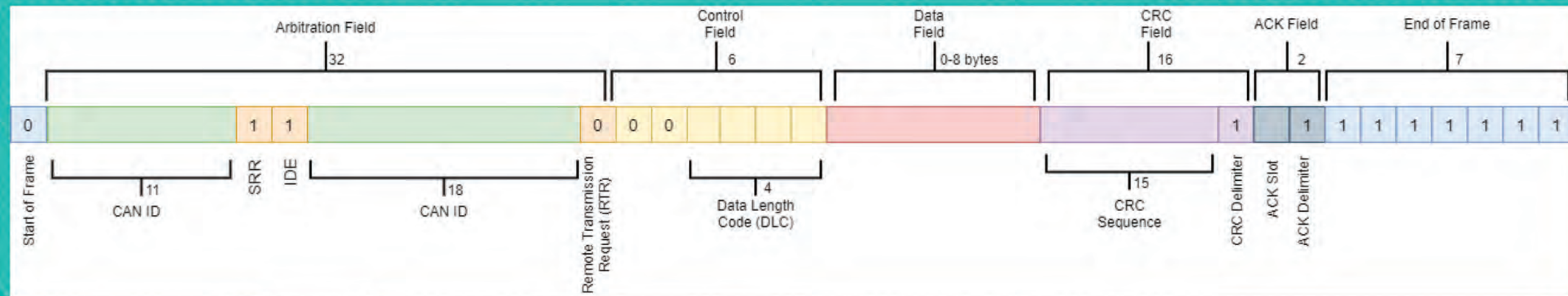
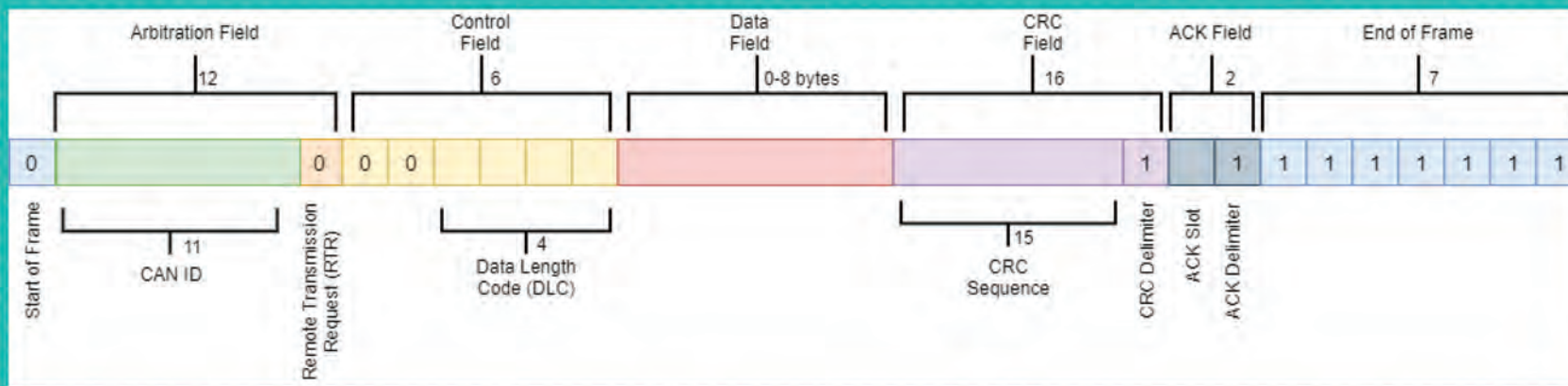
ACK field when transmitting data: 0b11
ACK field when transmitting the response: 0b01

CAN Data Frame – ACK Field

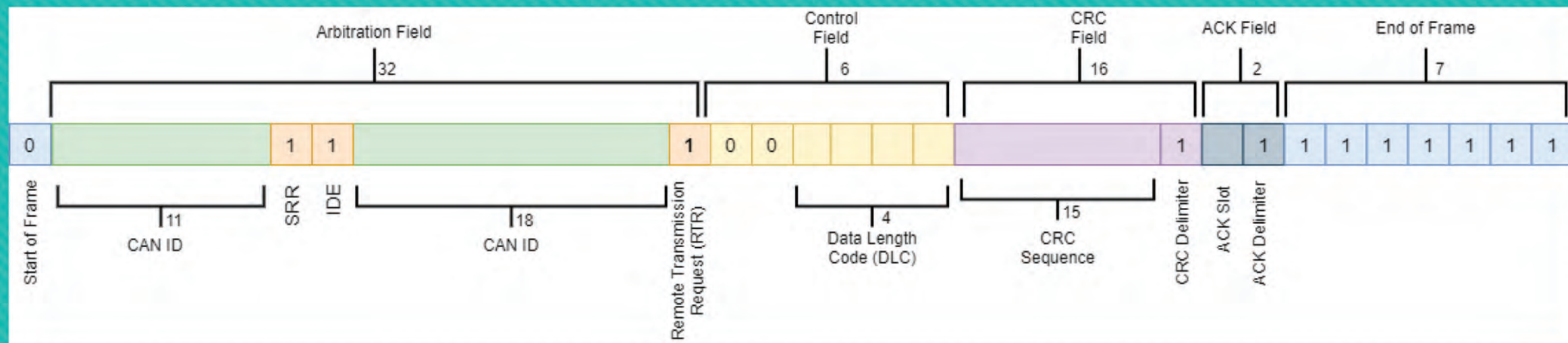




CAN Data Frame – Arbitration Field



CAN Data Frame – Extended Frame (CAN 2.0B)



CAN Remote Frame

Truck Networks

CAN (Controller Area Network)
- Physical and Data Link layers

J1939

- Higher layer protocol based on CAN
- Newer & faster

J1708

- Physical layer
- Primarily on older vehicles

J1587

- Higher layer protocol that runs on J1708

J2497

(PLC4TRUCKS)
- Basically J1708 but on trailer power lines



SAE J1939

Overview



CAN: PHYSICAL
AND DATA-LINK
LAYERS (ISO 11898)



MULTI-MASTER
DESIGN



NO ENCRYPTION



NO
AUTHENTICATION



EASY TO FLOOD
THE BUS

Common Attacks

Denial of
Service

Man in the
Middle

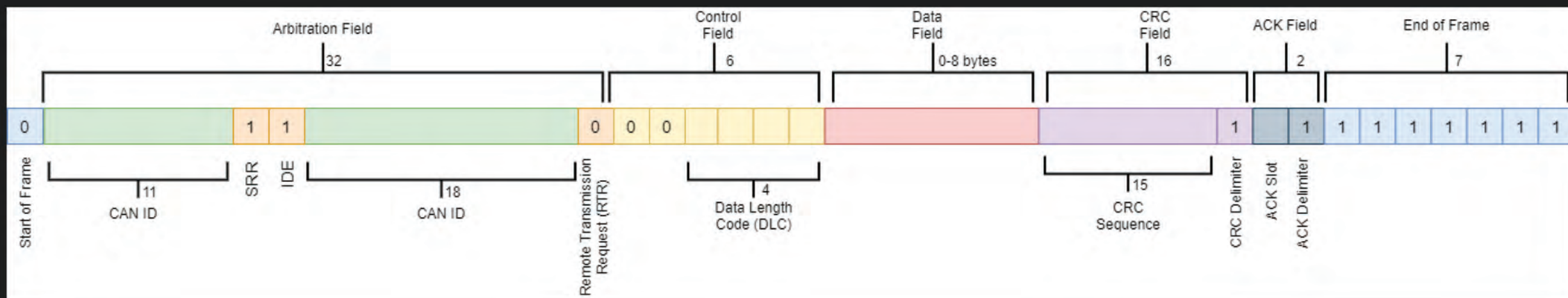
Diagnostic
protocol
abuse

Download
ECU firmware

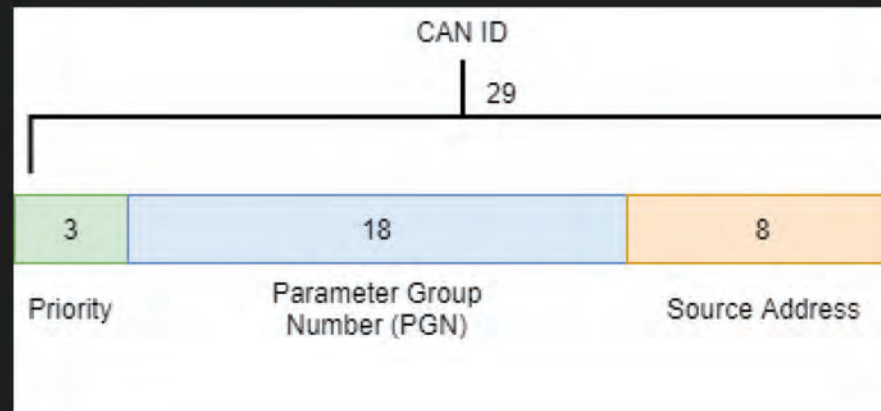
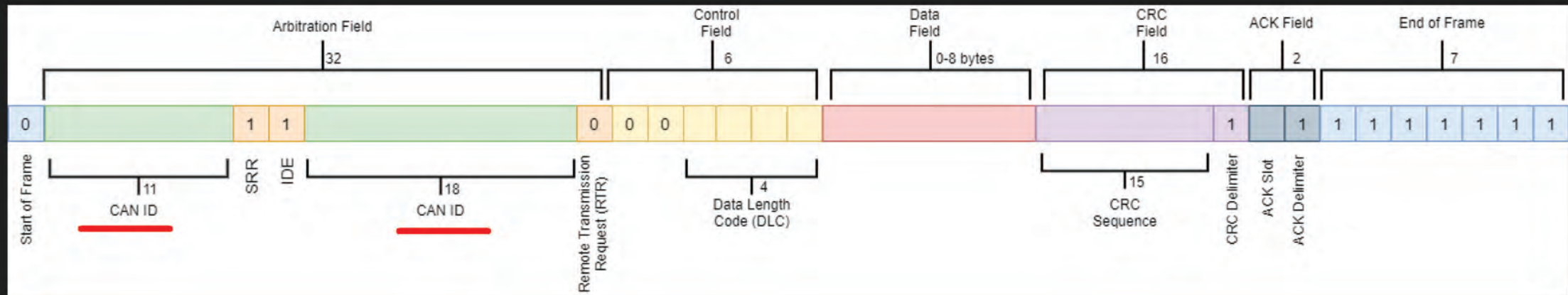
Reprogram
ECU

Fuzzing

J1939 Data Frame



J1939 CAN ID



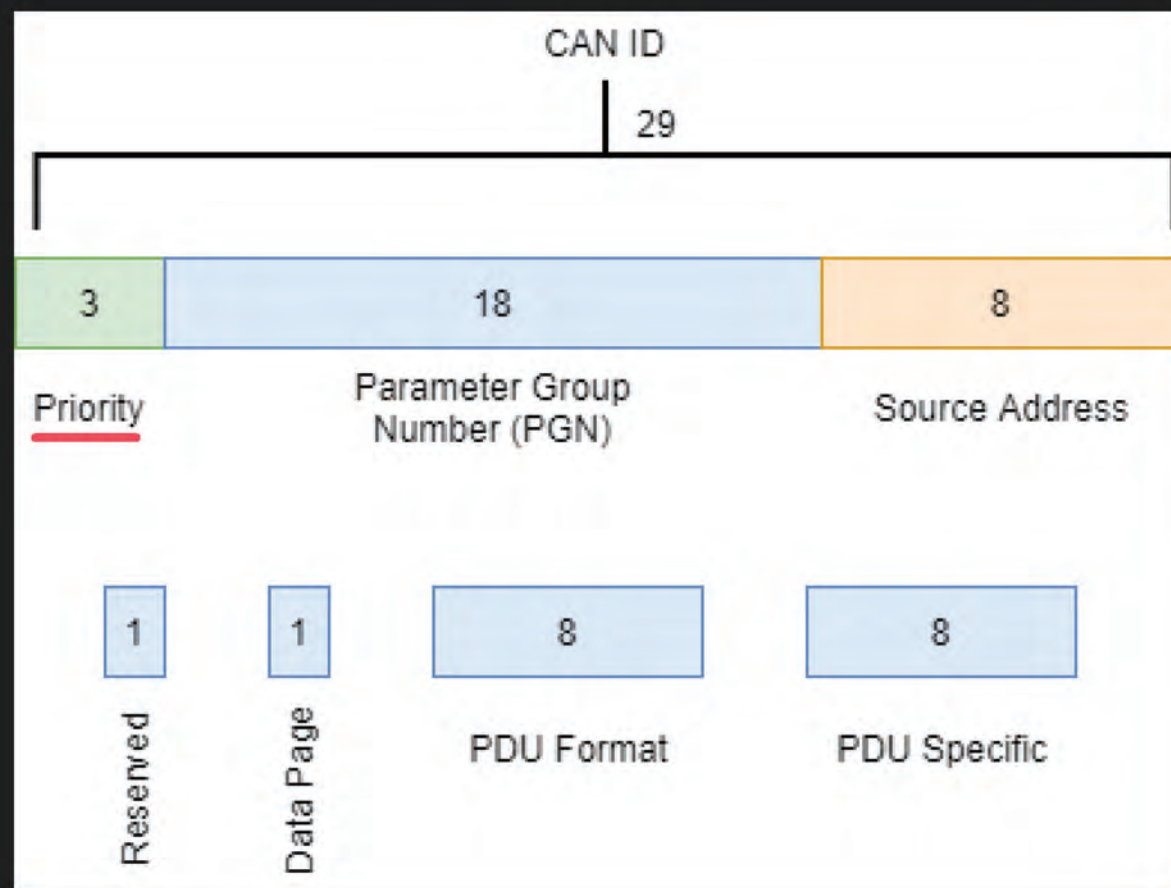
J1939 CAN ID – Priority

○ Priority:

0b000 – 0b111 (0-7)

0 = highest priority

7 = lowest priority



J1939 CAN ID – Source Address

Source Address:

0x00 – 0xFF (0-255)

Examples:

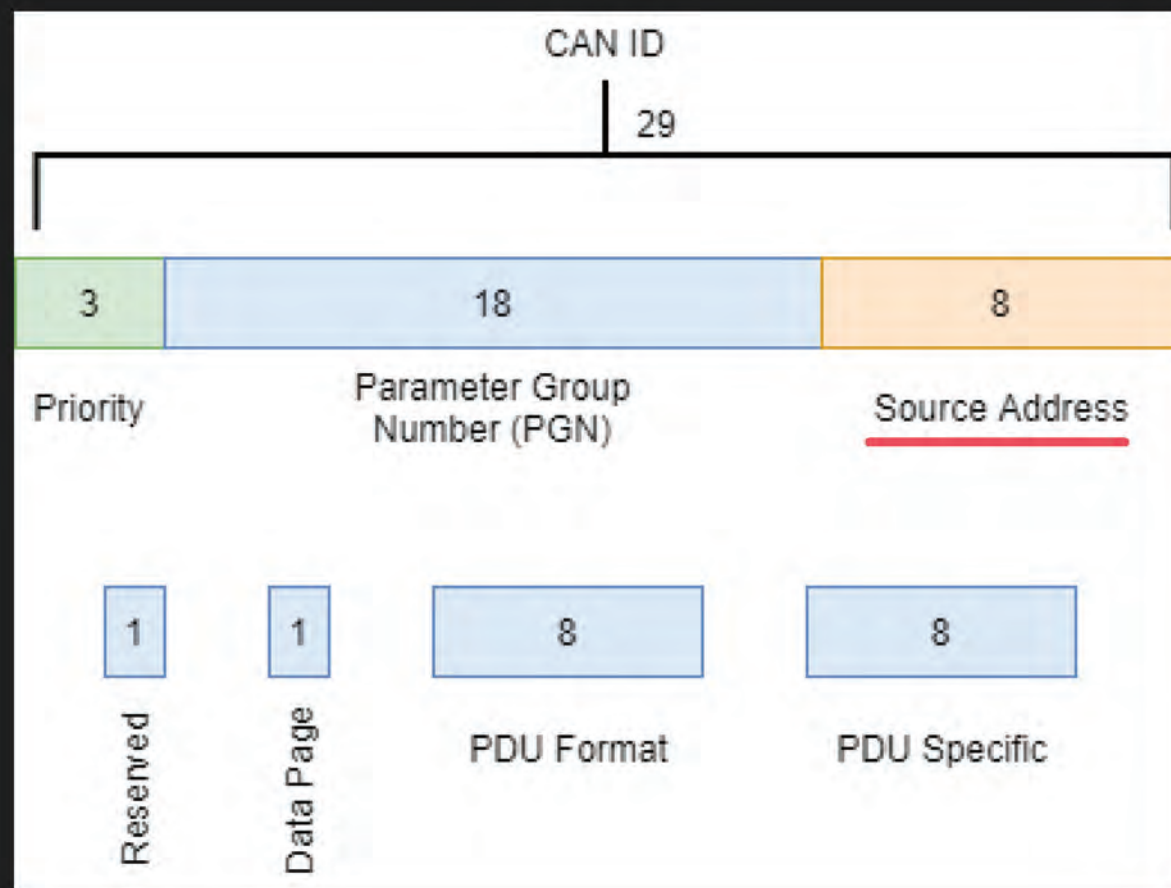
0 = Engine Control Module

11 = Brake Controller

249 = Diagnostic Device

254 = Reserved for Network Management

255 = Broadcast Message

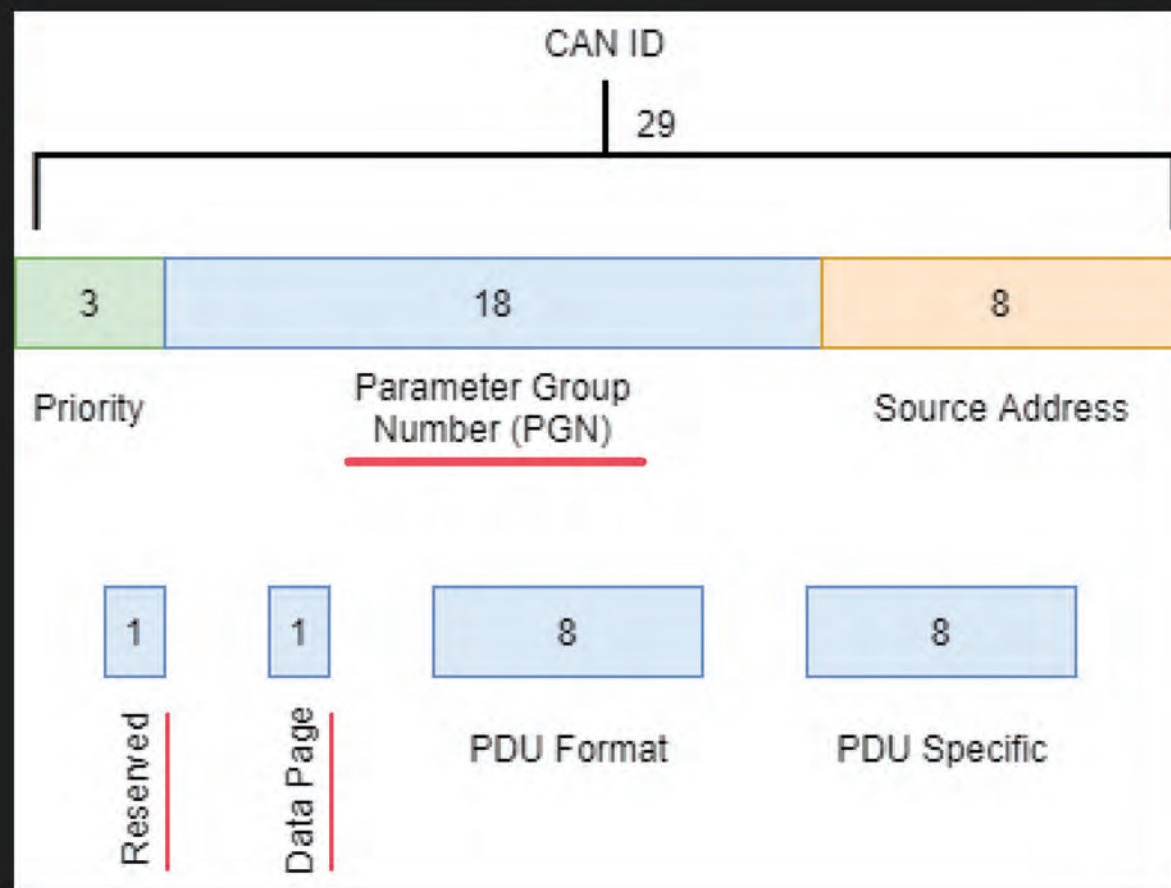


J1939 CAN ID – PGN

○Parameter Group Number (PGN)
Destination Specific or Broadcast

Reserved bit = 0

Data Page = 0



J1939 CAN ID – PGN

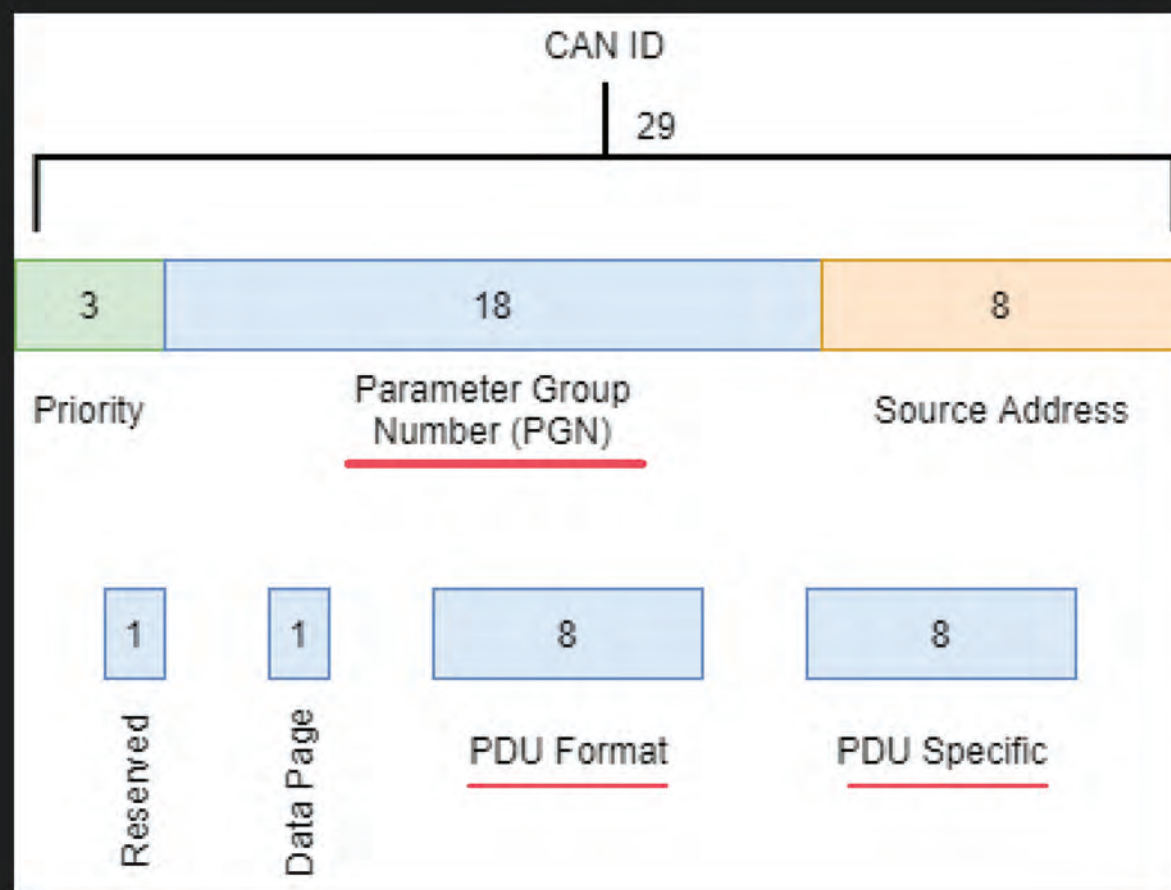
○ Destination Specific messages:

Node A -> Node B

PDU Format: 0-239

PDU Specific: 0-255 (destination address)

Parameter groups: 240



CAN ID Example #1 – Destination Specific

- General Purpose Valve Pressure

PDU Format: 0x07

PDU Specific: 0x0B (destination address)

Parameter Group: 0x0700

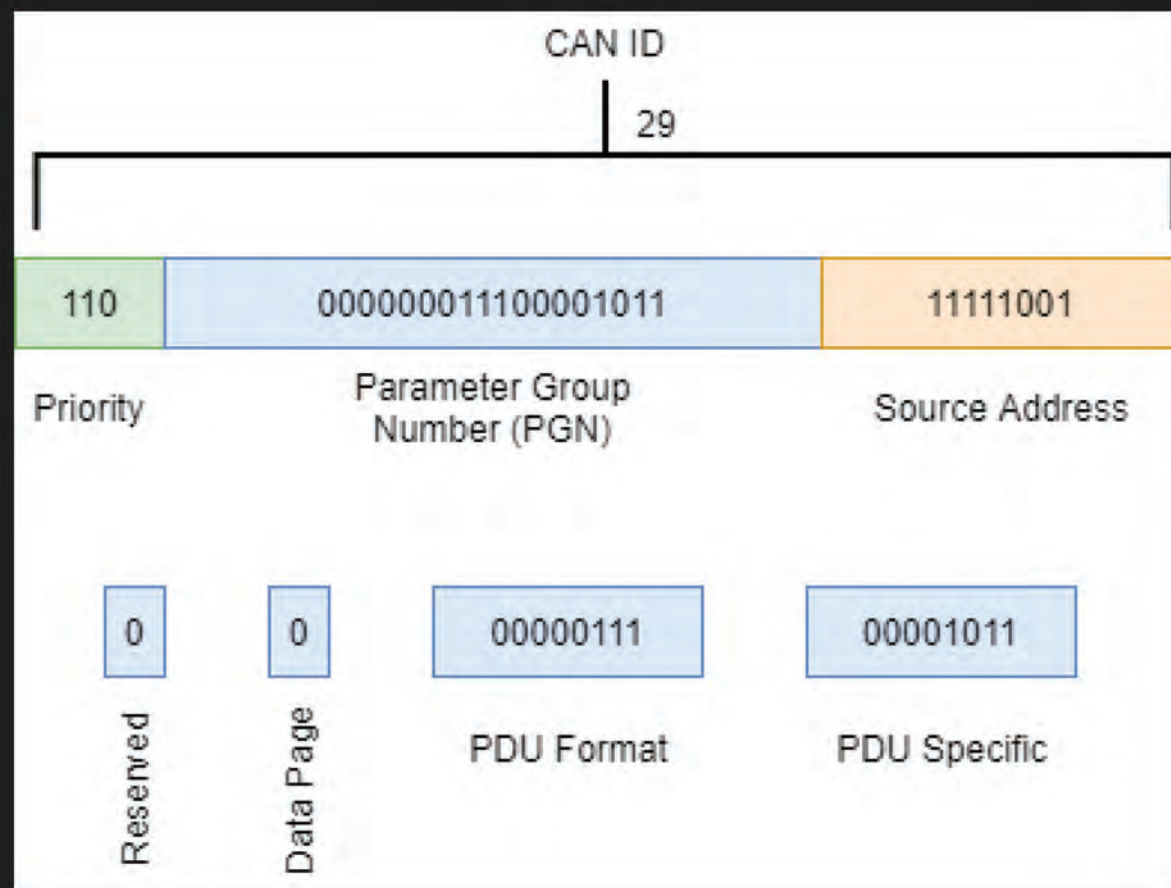
Priority: 6

Source Address: 0xF9

- CAN ID:

110 00 0000111 00001011 11111001

(0x18070BF9)



J1939 CAN ID – PGN

○ Broadcast messages:

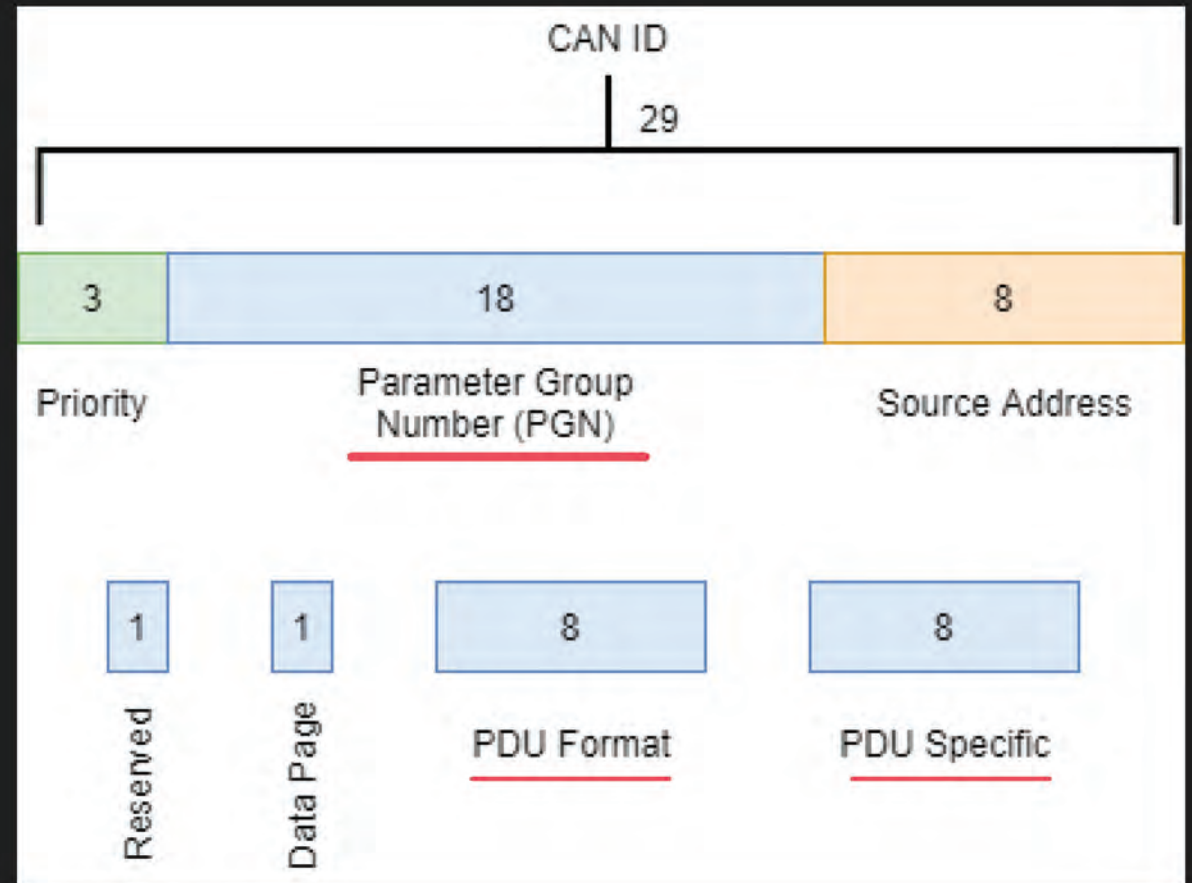
Node A -> All Nodes

PDU Format: 240-255

PDU Specific: 0-255 (group extension)

Parameter groups: 4096

Range: 61440-65535



CAN ID Example #2 - Broadcast

○ Electronic Engine Controller 3

PDU Format: 0xFE

PDU Specific: 0xDF

Parameter Group: 0xFEDF

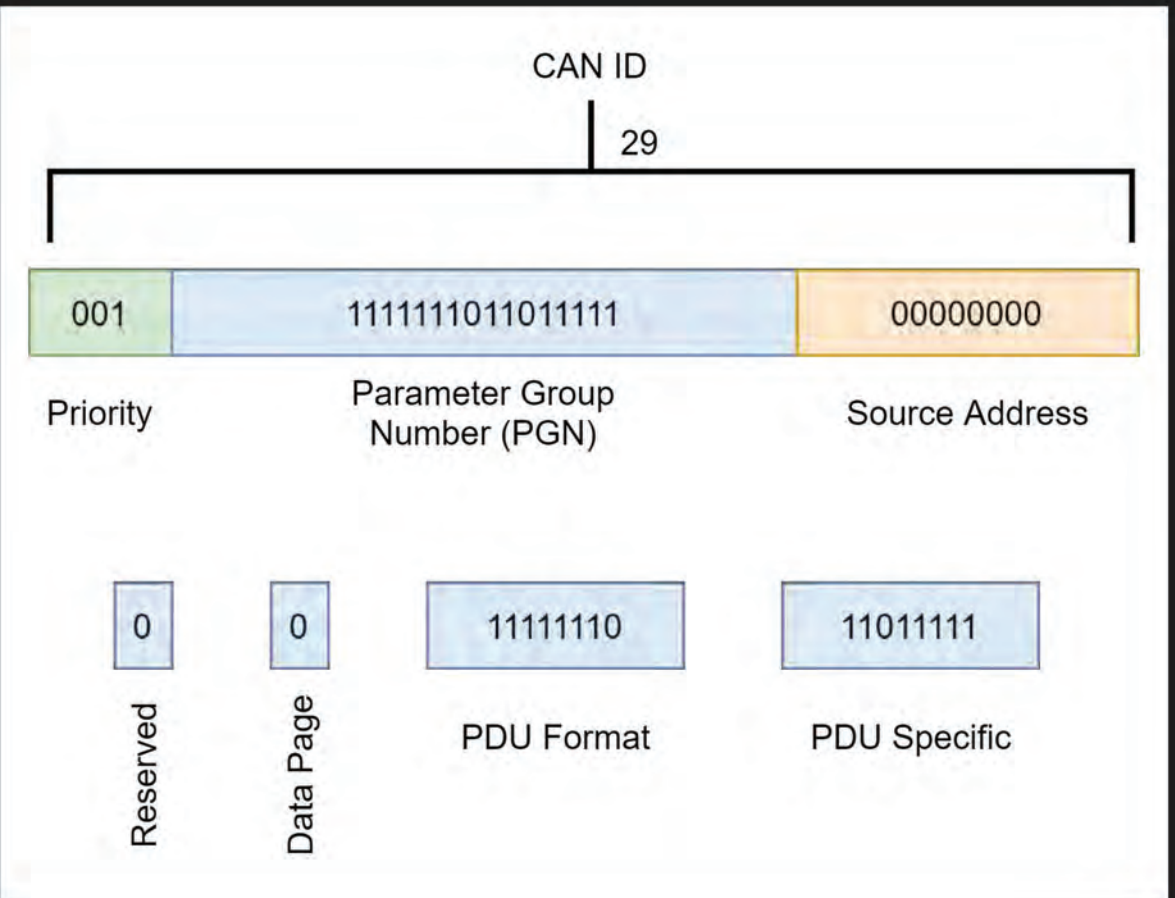
Priority: 1

Source Address: 0x00

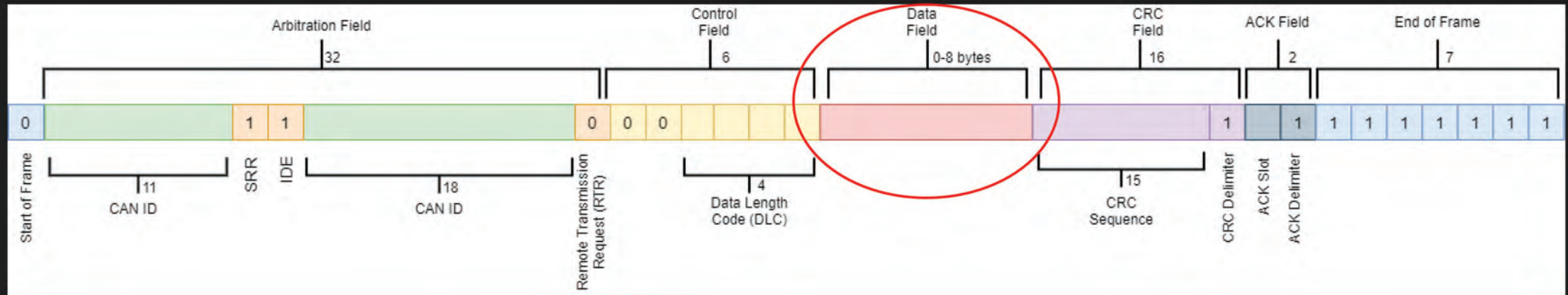
○ CAN ID:

001 00 11111110 11011111 00000000

(0x04FEDF00)

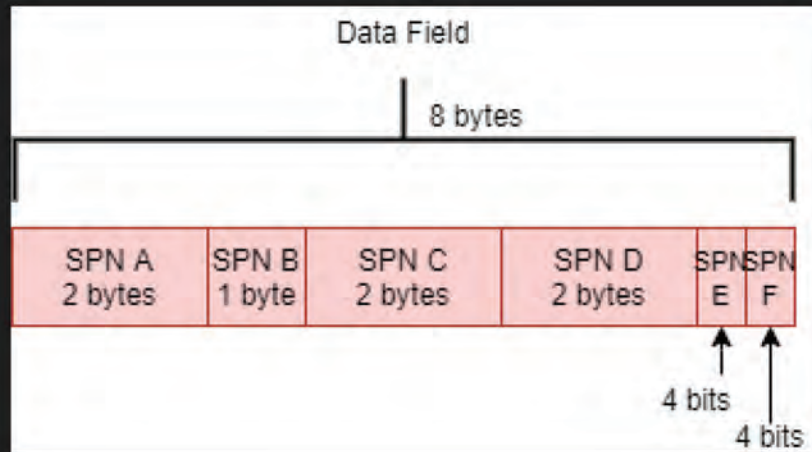


J1939 Data Field

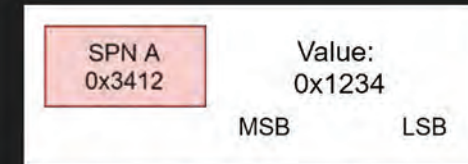


J1939 Data Field - Decoding

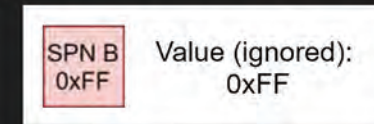
- Data field is decoded according to the SPNs that make up the message's associated PGN



- Each SPN is decoded least significant byte (LSB) first

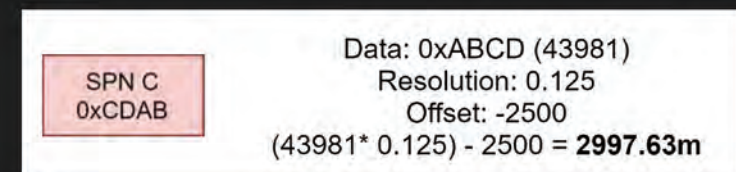


- SPNs containing all 1 bits can be ignored



- Calculated from SPN's resolution and offset

Real value = (Data * Resolution) + Offset



J1939 Data Field – Example #2

○ Electronic Engine Controller 3

PDU Format: 0xFE

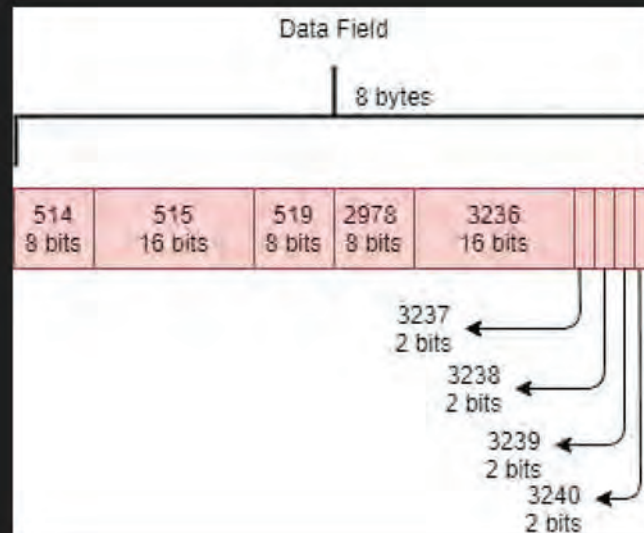
PDU Specific: 0xDF

Parameter Group: 0xFEDF (65247)

○ CAN ID: 0x18FEDF00

○ List of SPNs:

514, 515, 519, 2978, 3236,
3237, 3238, 3239, 3240



J1939 Data Field – Example #2

○ Electronic Engine Controller 3

PDU Format: 0xFE

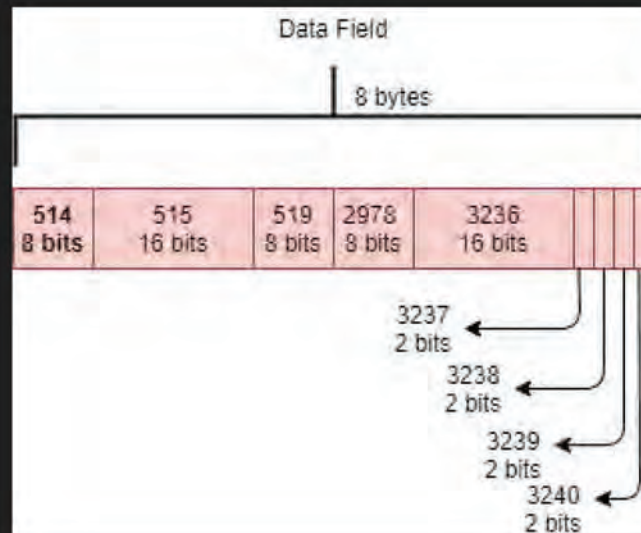
PDU Specific: 0xDF

Parameter Group: 0xFEDF (65247)

○ CAN ID: 0x18FEDF00

○ List of SPNs:

514, 515, 519, 2978, 3236,
3237, 3238, 3239, 3240



○ SPN 514 Details:

Name: Nominal Friction - Percent Torque

Bit Position: 0

Length: 1 byte

Resolution: 1

Offset: -125

Value: -125 - 125 %

○ Example:

Value: 0x81 (129)

$(129 * 1) - 125 = 4 \%$

J1939 Data Field – Example #2

○ Electronic Engine Controller 3

PDU Format: 0xFE

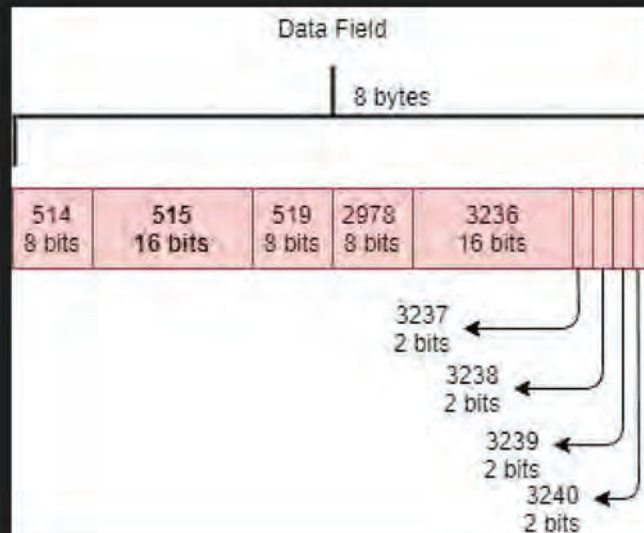
PDU Specific: 0xDF

Parameter Group: 0xFEDF (65247)

○ CAN ID: 0x18FEDF00

○ List of SPNs:

514, 515, 519, 2978, 3236,
3237, 3238, 3239, 3240



○ SPN 515 Details:

Name: Engine's Desired Operating Speed

Bit Position: 8

Length: 2 bytes

Resolution: 0.125

Offset: 0

Value: 0 - 8191 rpm

○ Example:

Value: 0xA028

LSB: 0x28A0 (10400)

$(10400 * 0.125) + 0 = 1300 \text{ rpm}$

J1939 Data Field – Example #2

○ Electronic Engine Controller 3

PDU Format: 0xFE

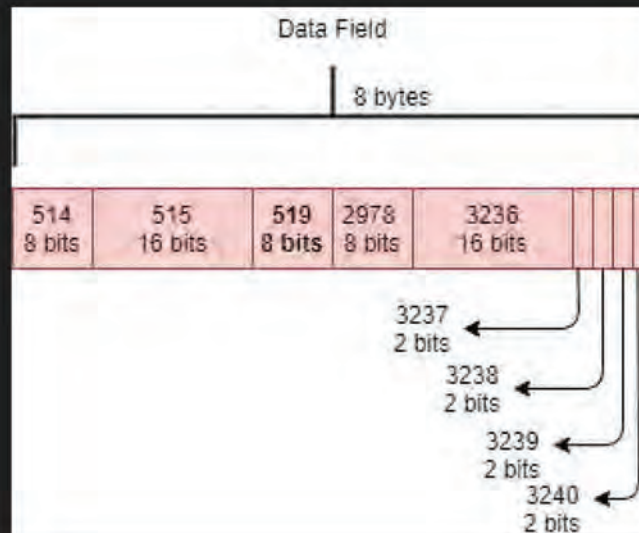
PDU Specific: 0xDF

Parameter Group: 0xFEDF (65247)

○ CAN ID: 0x18FEDF00

○ List of SPNs:

514, 515, 519, 2978, 3236,
3237, 3238, 3239, 3240



○ SPN 519 Details:

Name: Engine's Desired Operating Speed Asymmetry Adjustment

Bit Position: 24

Length: 1 byte

Resolution: 1

Offset: 0

Value: 0 - 250 ratio

○ Example:

Value: 0x7D

125 ratio

J1939 Data Field – Example #2

○ Electronic Engine Controller 3

PDU Format: 0xFE

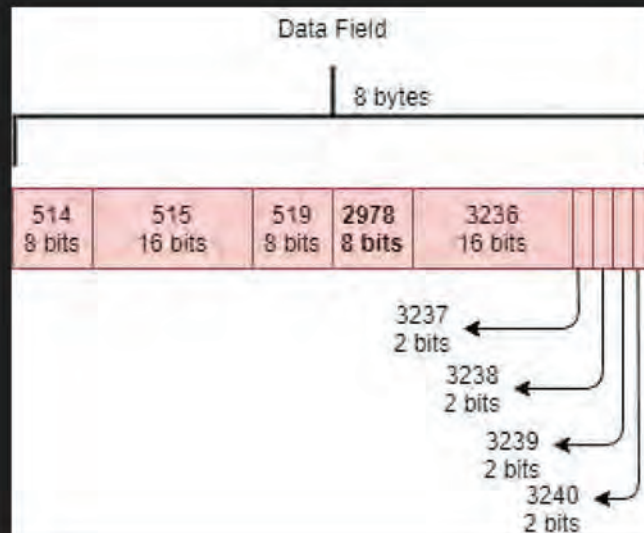
PDU Specific: 0xDF

Parameter Group: 0xFEDF (65247)

○ CAN ID: 0x18FEDF00

○ List of SPNs:

514, 515, 519, 2978, 3236,
3237, 3238, 3239, 3240



○ SPN 2978 Details:

Name: Estimated Engine Parasitic Losses - Percent Torque

Bit Position: 32

Length: 1 byte

Resolution: 1

Offset: -125

Value: -125 - 125 %

○ Example:

Value: 0xFB (251)

$(251 * 1) - 125 = 126 \%$

J1939 Data Field – Example #2

○ Electronic Engine Controller 3

PDU Format: 0xFE

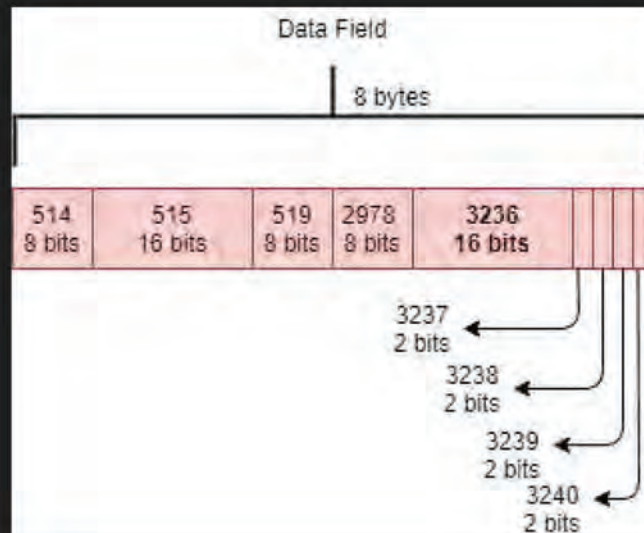
PDU Specific: 0xDF

Parameter Group: 0xFEDF (65247)

○ CAN ID: 0x18FEDF00

○ List of SPNs:

514, 515, 519, 2978, 3236,
3237, 3238, 3239, 3240



○ SPN 3236 Details:

Name: Aftertreatment 1 Exhaust Gas Mass Flow Rate

Bit Position: 40

Length: 2 bytes

Resolution: 0.2

Offset: 0

Value: 0 – 12851 kg/h

○ Example:

Value: 0x6C6B

LSB: 0x6B6C (27500)

$(27500 * 0.2) + 0 = 5500 \text{ kg/h}$

J1939 Data Field – Example #2

○ Electronic Engine Controller 3

PDU Format: 0xFE

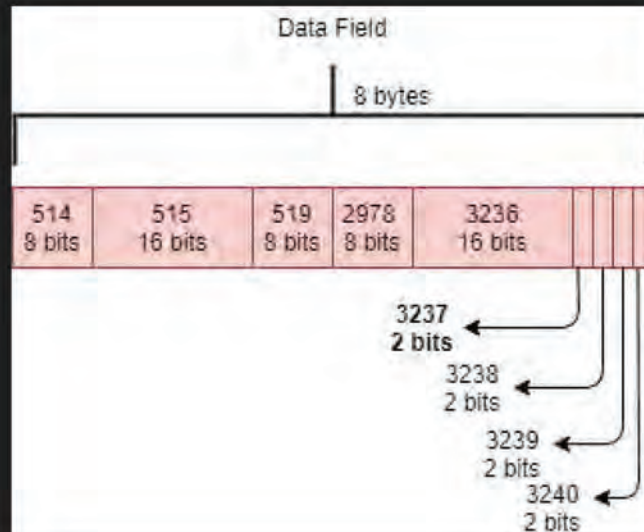
PDU Specific: 0xDF

Parameter Group: 0xFEDF (65247)

○ CAN ID: 0x18FEDF00

○ List of SPNs:

514, 515, 519, 2978, 3236,
3237, 3238, 3239, 3240



○ SPN 3237 Details:

Name: Aftertreatment 1 Intake Dew Point

Bit Position: 56

Length: 2 bits

Value: 0 – 3

○ To decode:

00 – not exceeded dew point

01 – exceeded dew point

10 – error

11 – ignore

J1939 Data Field – Example #2

○ Electronic Engine Controller 3

PDU Format: 0xFE

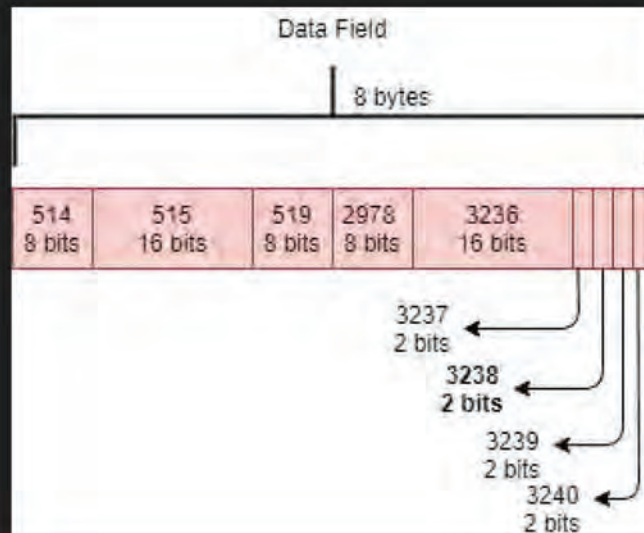
PDU Specific: 0xDF

Parameter Group: 0xFEDF (65247)

○ CAN ID: 0x18FEDF00

○ List of SPNs:

514, 515, 519, 2978, 3236,
3237, 3238, 3239, 3240



○ SPN 3238 Details:

Name: Aftertreatment 1 Exhaust Dew Point

Bit Position: 58

Length: 2 bits

Value: 0 – 3

○ To decode:

00 – not exceeded dew point

01 – exceeded dew point

10 – error

11 – ignore

J1939 Data Field – Example #2

○ Electronic Engine Controller 3

PDU Format: 0xFE

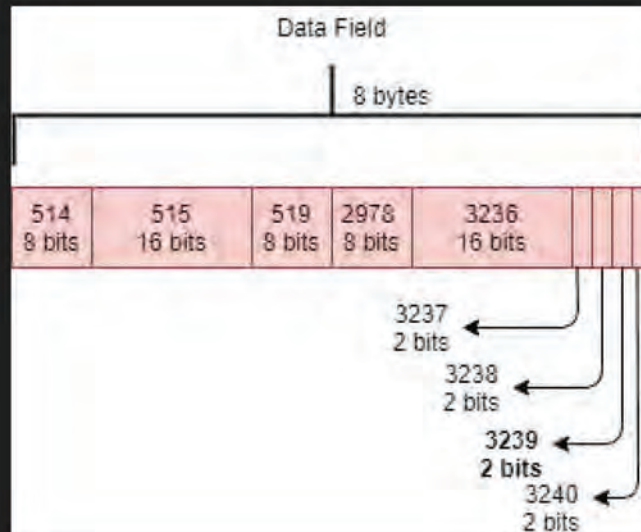
PDU Specific: 0xDF

Parameter Group: 0xFEDF (65247)

○ CAN ID: 0x18FEDF00

○ List of SPNs:

514, 515, 519, 2978, 3236,
3237, 3238, 3239, 3240



○ SPN 3239 Details:

Name: Aftertreatment 2 Intake Dew Point

Bit Position: 60

Length: 2 bits

Value: 0 – 3

○ To decode:

00 – not exceeded dew point

01 – exceeded dew point

10 – error

11 – ignore

J1939 Data Field – Example #2

○ Electronic Engine Controller 3

PDU Format: 0xFE

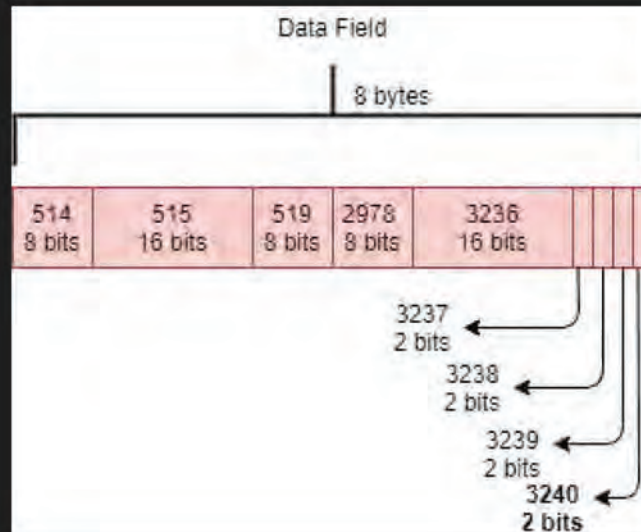
PDU Specific: 0xDF

Parameter Group: 0xFEDF (65247)

○ CAN ID: 0x18FEDF00

○ List of SPNs:

514, 515, 519, 2978, 3236,
3237, 3238, 3239, 3240



○ SPN 3240 Details:

Name: Aftertreatment 2 Intake Dew Point

Bit Position: 62

Length: 2 bits

Value: 0 – 3

○ To decode:

00 – not exceeded dew point

01 – exceeded dew point

10 – error

11 – ignore

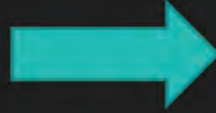
J1939 Data Field – Example #2

○ Electronic Engine Controller 3

PDU Format: 0xFE

PDU Specific: 0xDF

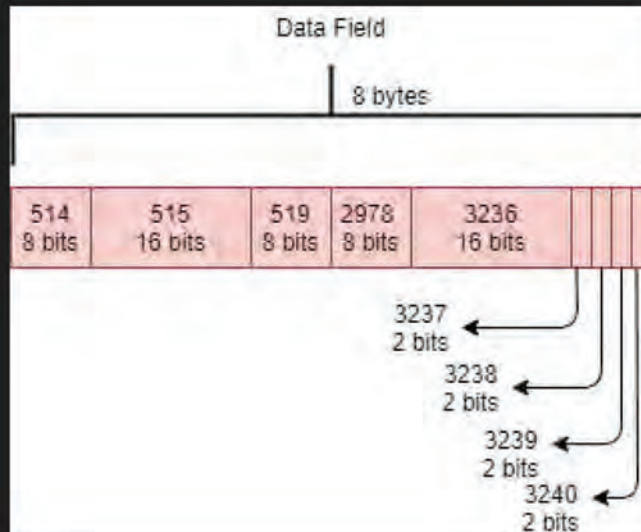
Parameter Group: 0xFEDF (65247)



○ CAN ID: 0x18FEDF00

○ List of SPNs:

514, 515, 519, 2978, 3236,
3237, 3238, 3239, 3240



○ Data Field:

514 (0x81) +

515 (0xA028) +

519 (0x7D) +

2978 (0xFB) +

3236 (0x6C6B) +

3237/3238/3239/3240

11 11 00 01 (0xF1) =

0x81A0287DFB6C6BF1



TruckDevil

<https://github.com/LittleBlondeDevil/TruckDevil>

Demo 1: reading messages

Demo 1: readMessages.py

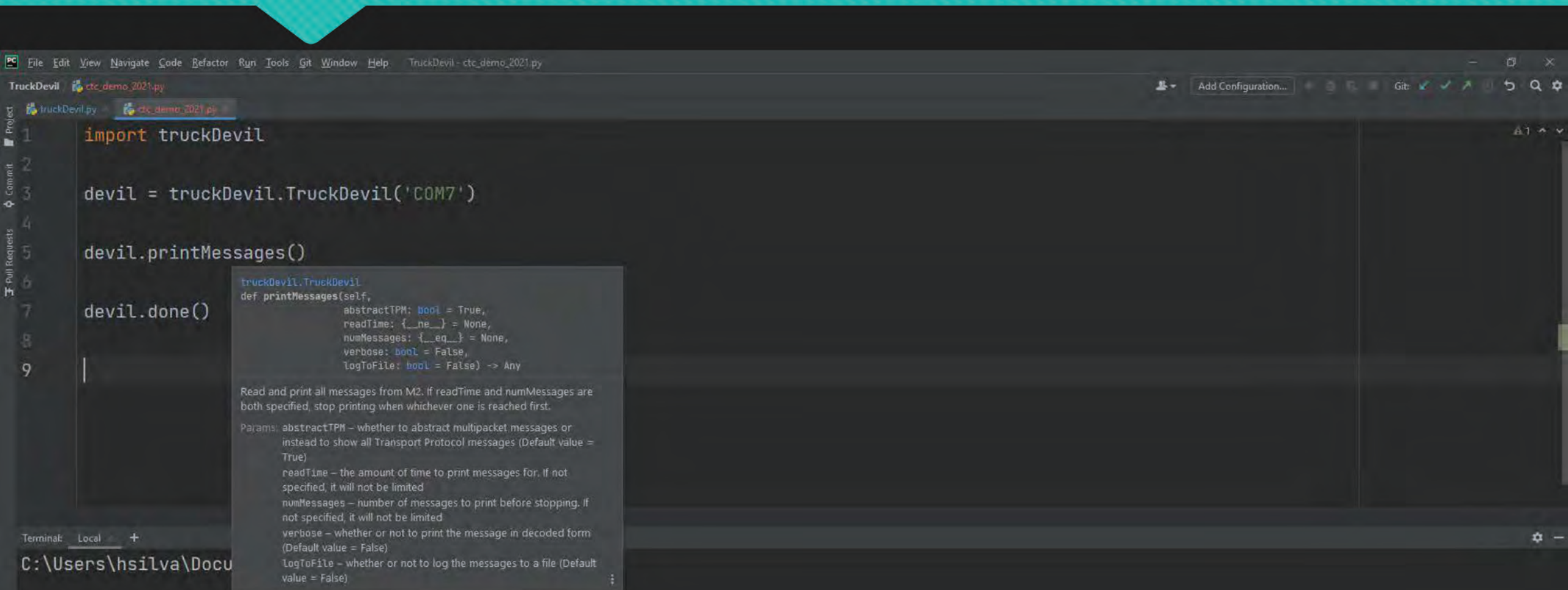
```
PS C:\Users\hsilva\Documents\TruckDevil> python readMessages.py -h
usage: readMessages.py [-h] [-s SERIAL_BAUD] [-t READ_TIME] [-n NUM_MESSAGES]
                        [-a] [-l] [-v]
                        port can_baud

read and print all messages from M2. If read_time and num_messages are both
specified, stop printing when whichever one is reached first.

positional arguments:
  port                  serial port that the M2 is connected to. For example:
                        COM7 or /dev/ttyX.
  can_baud              baud rate on the CAN BUS that the M2 is connected. For
                        example: 250000.

optional arguments:
  -h, --help            show this help message and exit
  -s SERIAL_BAUD, --serial_baud SERIAL_BAUD
                        baud rate of the serial connection to the M2. By
                        default it is 115200.
  -t READ_TIME, --read_time READ_TIME
                        the amount of time, in seconds, to print messages for.
                        If not specified, it will not be limited.
  -n NUM_MESSAGES, --num_messages NUM_MESSAGES
                        number of messages to print before stopping. If not
                        specified, it will not be limited.
  -a, --abstract_TPM    abstract Transport Protocol messages.
  -l, --log_to_file      log the messages to a file in the current directory
                        with the form 'm2_collected_data_[TIME]'.
  -v, --verbose          print the message in decoded form
```


Demo 1: printMessages()



The screenshot shows an IDE window titled "TruckDevil - ctc_demo_2021.py". The editor displays a Python script with the following code:

```
1 import truckDevil
2
3 devil = truckDevil.TruckDevil('COM7')
4
5 devil.printMessages()
6
7 devil.done()
8
9
```

A tooltip or documentation window is open over the `printMessages` method, showing its signature and parameters:

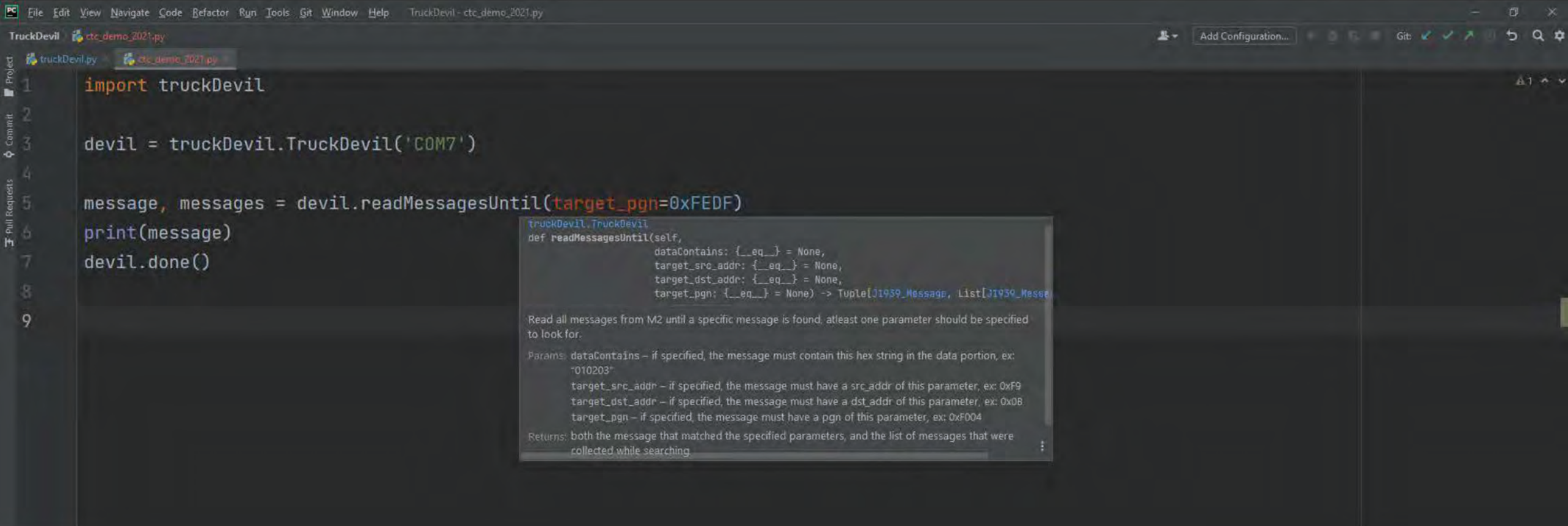
```
truckDevil.TruckDevil
def printMessages(self,
    abstractTPM: bool = True,
    readTime: {__ne__} = None,
    numMessages: {__eq__} = None,
    verbose: bool = False,
    logToFile: bool = False) -> Any
```

Read and print all messages from M2. If readTime and numMessages are both specified, stop printing when whichever one is reached first.

Params: abstractTPM – whether to abstract multipacket messages or instead to show all Transport Protocol messages (Default value = True)
readTime – the amount of time to print messages for. If not specified, it will not be limited
numMessages – number of messages to print before stopping. If not specified, it will not be limited
verbose – whether or not to print the message in decoded form (Default value = False)
logToFile – whether or not to log the messages to a file (Default value = False)

The bottom of the IDE shows a terminal window with the path `C:\Users\hsilva\Docu`.

Demo 1: readMessagesUntil()



The screenshot shows a code editor with a Python script and a tooltip for the `readMessagesUntil()` method.

Code Editor:

```
1 import truckDevil
2
3 devil = truckDevil.TruckDevil('COM7')
4
5 message, messages = devil.readMessagesUntil(target_pgn=0xFEDF)
6 print(message)
7 devil.done()
8
9
```

Tooltip:

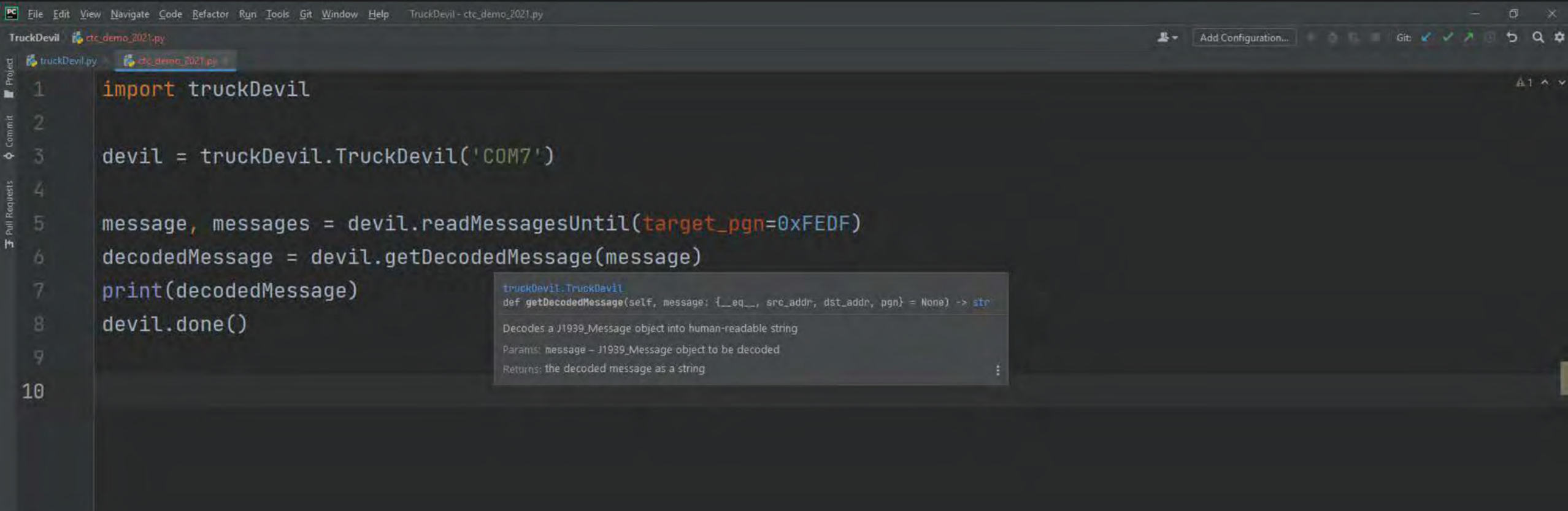
`truckDevil.TruckDevil`
`def readMessagesUntil(self,`
 `dataContains: {__eq__} = None,`
 `target_src_addr: {__eq__} = None,`
 `target_dst_addr: {__eq__} = None,`
 `target_pgn: {__eq__} = None) -> Tuple[J1939_Message, List[J1939_Message]]`

Read all messages from M2 until a specific message is found, atleast one parameter should be specified to look for.

Params: `dataContains` – if specified, the message must contain this hex string in the data portion, ex: "010203"
 `target_src_addr` – if specified, the message must have a `src_addr` of this parameter, ex: 0xF9
 `target_dst_addr` – if specified, the message must have a `dst_addr` of this parameter, ex: 0x0B
 `target_pgn` – if specified, the message must have a `pgn` of this parameter, ex: 0xF004

Returns: both the message that matched the specified parameters, and the list of messages that were collected while searching

Demo 1: getDecodedMessage()



The screenshot shows a Python IDE with a dark theme. The main editor window displays a script named `ctc_demo_2021.py` with the following code:

```
1 import truckDevil
2
3 devil = truckDevil.TruckDevil('COM7')
4
5 message, messages = devil.readMessagesUntil(target_pgn=0xFEDF)
6 decodedMessage = devil.getDecodedMessage(message)
7 print(decodedMessage)
8 devil.done()
9
10
```

A tooltip is visible over the `getDecodedMessage` method call on line 6. The tooltip contains the following information:

- Method signature: `truckDevil.TruckDevil`
`def getDecodedMessage(self, message: {__eq__, src_addr, dst_addr, pgn} = None) -> str`
- Description: Decodes a J1939_Message object into human-readable string
- Params: message – J1939_Message object to be decoded
- Returns: the decoded message as a string

Demo 2: sending messages

Be careful!

Demo 2: sendMessage.py

```
PS C:\Users\hsilva\Documents\TruckDevil> python sendMessage.py -h
usage: sendMessage.py [-h] [-s SERIAL_BAUD] [-p PRIORITY] [-a SRC_ADDR]
                    [-d DST_ADDR] [-v]
                    port can_baud pgn data

send message to M2 to get pushed to the BUS.

positional arguments:
  port                serial port that the M2 is connected to. For example:
                     COM7 or /dev/ttyX.
  can_baud            baud rate on the CAN BUS that the M2 is connected. For
                     example: 250000.
  pgn                 range: 0x0000-0xFFFF (0-65535).
  data                hex string of data to send, example: 0102030405060708.

optional arguments:
  -h, --help          show this help message and exit
  -s SERIAL_BAUD, --serial_baud SERIAL_BAUD
                     baud rate of the serial connection to the M2. Default:
                     115200.
  -p PRIORITY, --priority PRIORITY
                     range: 0x00-0x07 (0-7).
  -a SRC_ADDR, --src_addr SRC_ADDR
                     range: 0x00-0xFF (0-255).
  -d DST_ADDR, --dst_addr DST_ADDR
                     range: 0x00-0xFF (0-255), 0xFF is for broadcast
                     messages.
  -v, --verbose       print the message that was sent, use -vv to print the
                     decoded form of the message.
```

Demo 2: J1939_Message object

TruckDevil - ctc_demo_2021.py

TruckDevil - ctc_demo_2021.py

truckDevil.py ctc_demo_2021.py

1
2
3
4
5
6
7
8
9
10

```
import truckDevil

devil = truckDevil.TruckDevil('COM7')

message = truckDevil.J1939_Message(0, 0xFEDF, 0xFF, 0x00, "81A0287DFB6C6BF1")
decodedMessage = devil.get_message(message)
print(decodedMessage)
devil.done()
```

```
truckDevil.J1939_Message
def __init__(self,
    priority: int = 0x00,
    pgn: int = 0x0000,
    dst_addr: int = 0xFF,
    src_addr: int = 0x00,
    data: str = "0000000000000000",
    total_bytes: {__eq__, __lt__} = None) -> Any
```

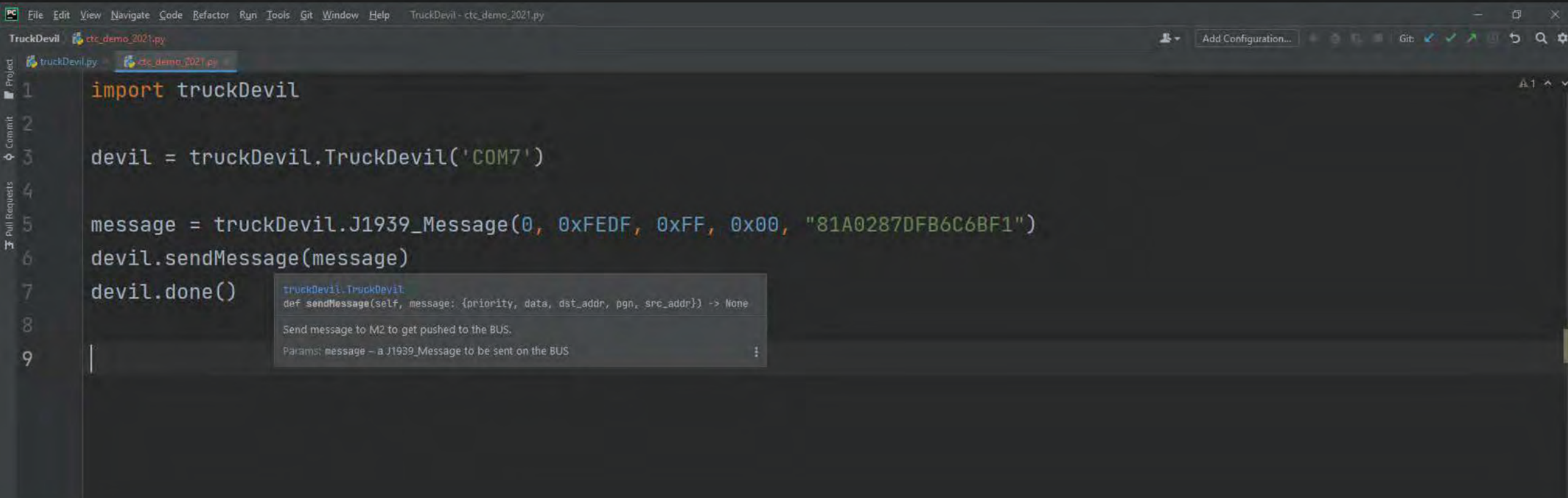
Data object for storing the contents of a single J1939 message

Params:

- priority - 0x00-0x07 (Default value = 0x00)
- pgn - 0x0000-0xFFFF (Default value = 0x0000)
- dst_addr - 0x00-0xFF, 0xFF is for broadcast (Default value = 0xFF)
- src_addr - 0x00-0xFF (Default value = 0x00)
- data - hex string, eg: "0102030405060708" (Default value = "0000000000000000")
- total_bytes - >= 0

Documentation is copied from: [J1939_Message](#)

Demo 2: sendMessage()



```
TruckDevil - ctc_demo_2021.py
TruckDevil ctc_demo_2021.py
1 import truckDevil
2
3 devil = truckDevil.TruckDevil('COM7')
4
5 message = truckDevil.J1939_Message(0, 0xFEDF, 0xFF, 0x00, "81A0287DFB6C6BF1")
6 devil.sendMessage(message)
7 devil.done()
8
9
```

truckDevil.TruckDevil
def sendMessage(self, message: {priority, data, dst_addr, pgn, src_addr}) -> None
Send message to M2 to get pushed to the BUS.
Params: message – a J1939_Message to be sent on the BUS

Challenge 1

- Find and decode message from the bus with PGN 0xF004 from ECU with source address 0x00
- Add +50% to SPN 513 (Actual Engine – Percent Torque)
- Resend the modified message, spoofing a source address of 0x00

Interesting J1939 Messages

J1939 Request Message

- Diagnostic Device requests ECU Identification Information from the Brake Controller

- Request

PDU Format: 0xEA

PDU Specific: 0x0B (destination address)

Parameter Group: 0xEA00

Priority: 6

Source Address: 0xF9

- CAN ID:

110 00 11101010 00001011 11111001

(0x18EA0BF9)

- Contains only 1 SPN: 2540

- SPN 2540 Details:

Name: Parameter Group Number (RQST)

Bit Position: 0

Length: 3 bytes

Value: 0x00000 – 0xFFFFFFFF

- To request ECU Identification Information (0x00FDC5):

Data field: 0xC5FD00

J1939 Acknowledgement Message

- Brake Controller acknowledges the receipt of the Request message

- Acknowledgement Message

PDU Format: 0xE8

PDU Specific: 0xFF (global address)

Parameter Group: 0xE800

Priority: 6

Source Address: 0x0B

- CAN ID:

110 00 11101000 11111111 00001011

(0x18E8FF0B)

- Data field:

Byte 1:

0x00 – positive ack

0x01 – negative ack

0x02 – access denied

0x03 – cannot respond

Byte 2: Group Function Value

Byte 3-4: 0xFFFF

Byte 5: source address of originating request message

Byte 6-8: PGN of requested data

- To acknowledge request from Diagnostic Device:

Byte 1: 0x00

Bytes 2-4: 0xFFFFFFFF

Byte 5: 0xF9

Bytes 6-8: 0xC5FD00

Data Field:

0x00FFFFFFFFF9C5FD00



Demo 3: sending Request message

J1939 Proprietary Messages

- Manufacturers can create their own messages
- The data field cannot be parsed without reverse engineering the proprietary protocol

- Proprietary A

PDU Format: 0xEF

PDU Specific: destination address

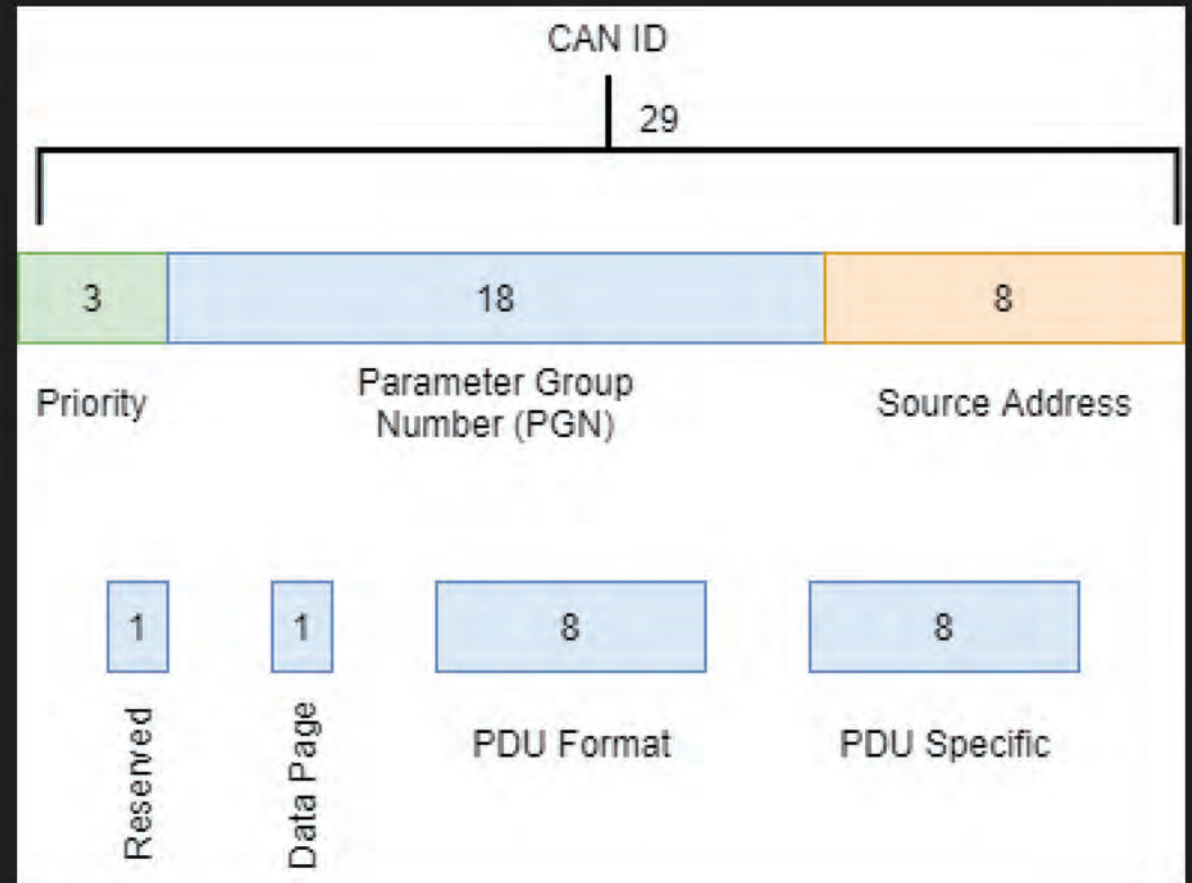
Parameter Group: 0xEF00

- Proprietary B

PDU Format: 0xFF

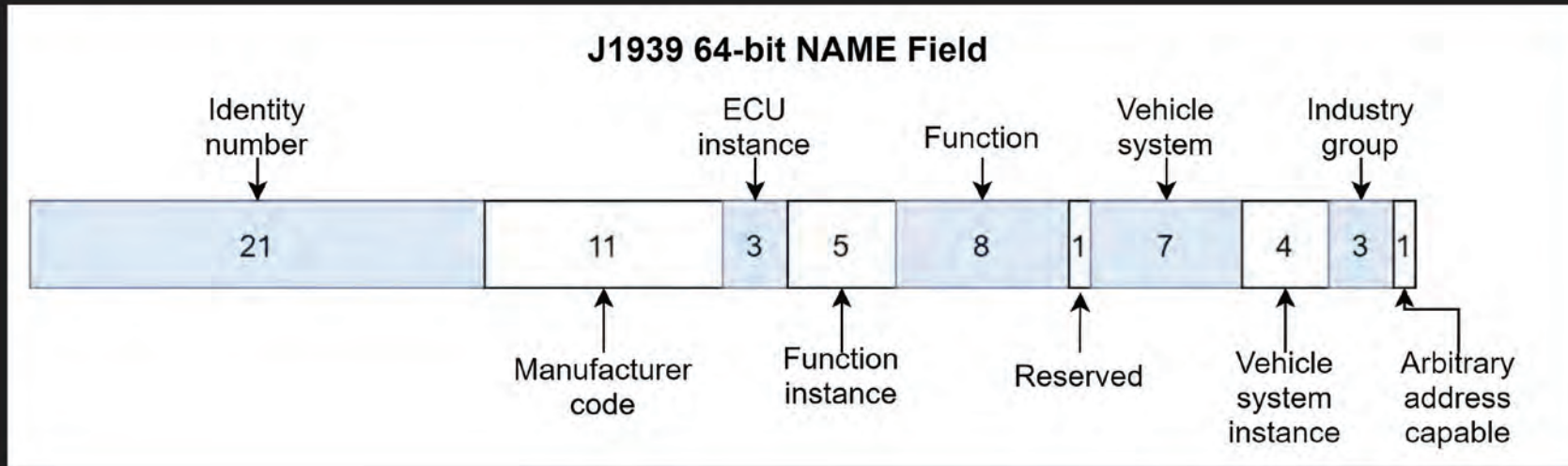
PDU Specific: 0x00-0xFF

Parameter Groups: 0xFF00-0xFFFF



J1939 Network Management - NAME

- Each ECU, or node, on the network must have a NAME and a source address
- NAME is 8 bytes and describes the ECU's function



J1939 Network Management - Address Claim

- The SAE standard states that ECUs should claim an address before sending messages on the network

- Address Claim:

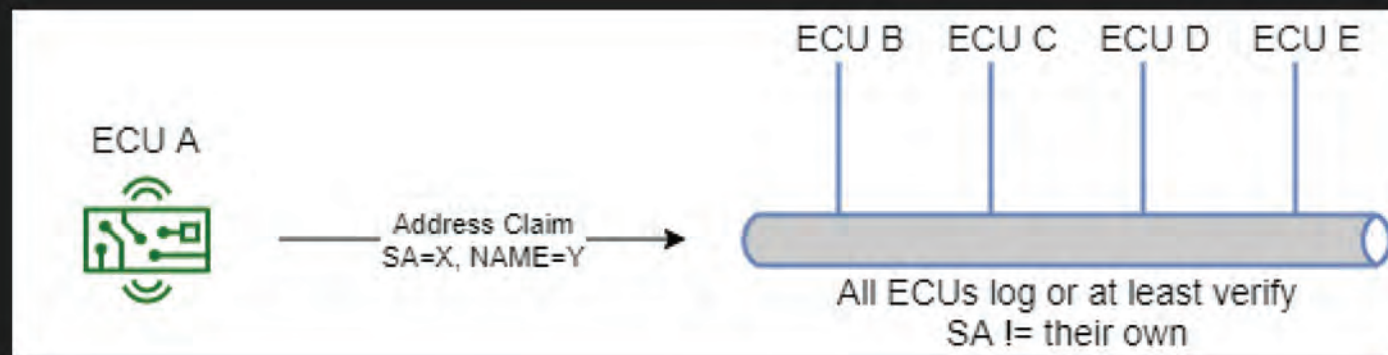
PDU Format: 0xEE

PDU Specific: 0xFF (global destination address)

Parameter Group: 0xEE00

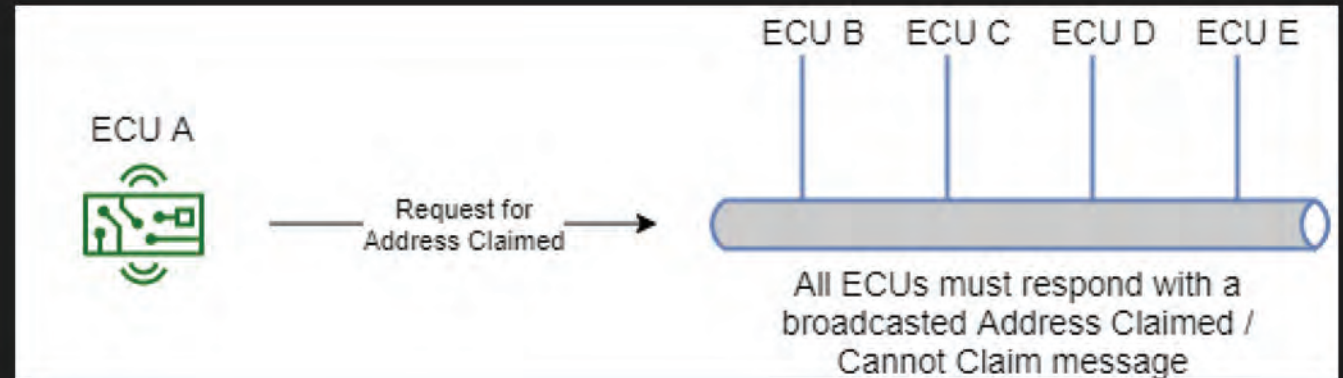
Source Address contains the one being claimed

Data: 8 bytes containing the NAME



J1939 Network Management – Request AC

- The SAE standard states that ECUs should claim an address before sending messages on the network
- Request for Address Claimed:
Request message (PGN 0xEA00)
PGN in Data Field is 0xEE00
- Cannot Claim Address:
Same as Address Claim but SA=0xFE



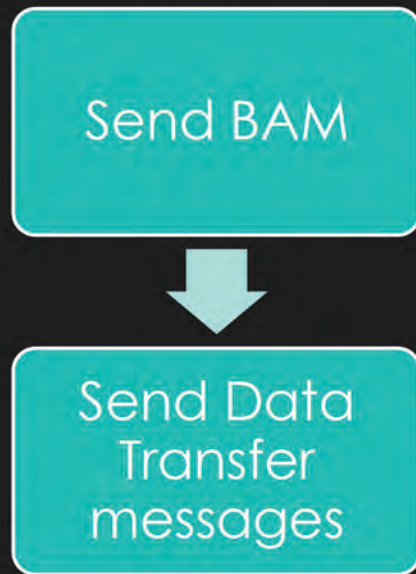


Demo 4: request Address Claimed

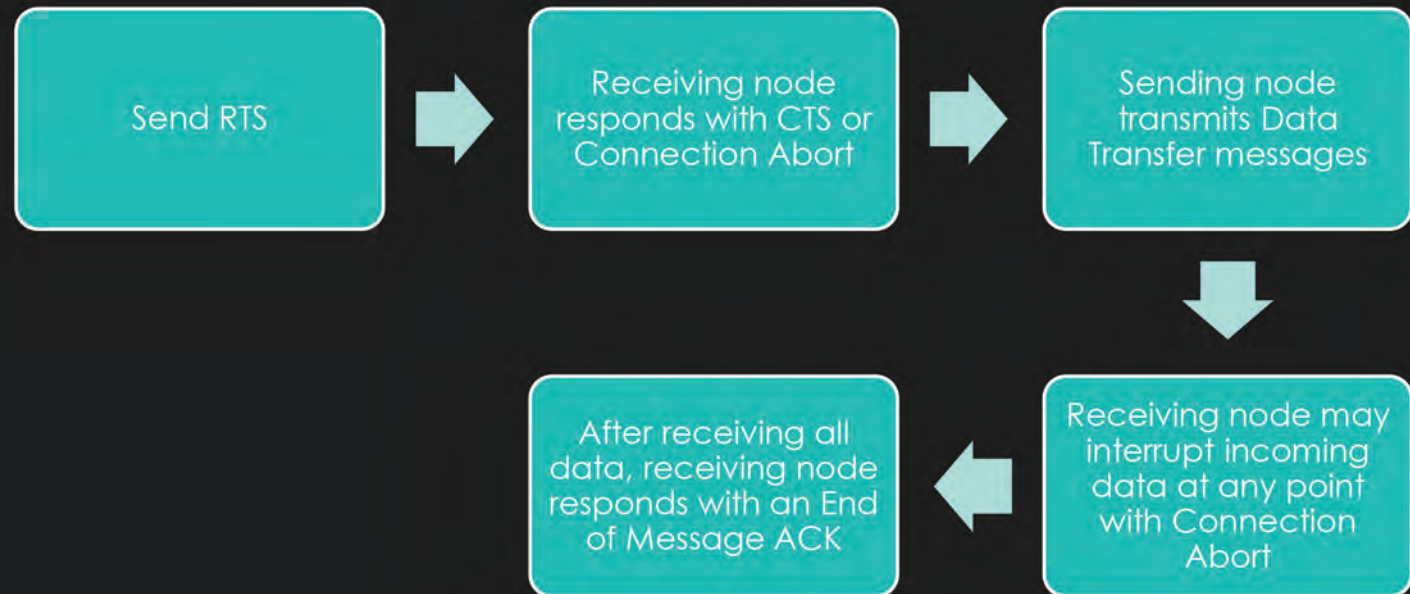
J1939 Transport Protocol

Multi-packet messages

Broadcast Messages



Destination Specific Message



Multi-packet Broadcast Message – BAM

- Transport Protocol – Connection Mgmt

PDU Format: 0xEC

PDU Specific: 0xFF (global address)

Parameter Group: 0xEC00

Priority: 6

Source Address: 0x00

- CAN ID:

110 00 11101100 11111111 00000000

(0x18ECFF00)

- Data field:

Byte 1: 0x20

Byte 2-3: message size, in bytes

Byte 4: number of packets needed to send entire multi-packet message

Byte 5: 0xFF

Byte 6-8: PGN of the multi-packet message

- Example BAM message:

PGN 0xFECA, 18 bytes long, sent over 3 packets

0x 20 1200 03 FF CAFE00

Multi-packet Broadcast Message – Data Transfer

- Transport Protocol – Data Transfer

PDU Format: 0xEB

PDU Specific: 0xFF (global address)

Parameter Group: 0xEB00

Priority: 6

Source Address: 0x00

- CAN ID:

110 00 11101011 11111111 00000000

(0x18EBFF00)

- Data field:

Byte 1: 0x01-0xFF (sequence number)

Bytes 2-8: the data

- Example Data Transfer:

Packet 1: 0x0101020304050607

Packet 2: 0x0208090A0B0C0D0E

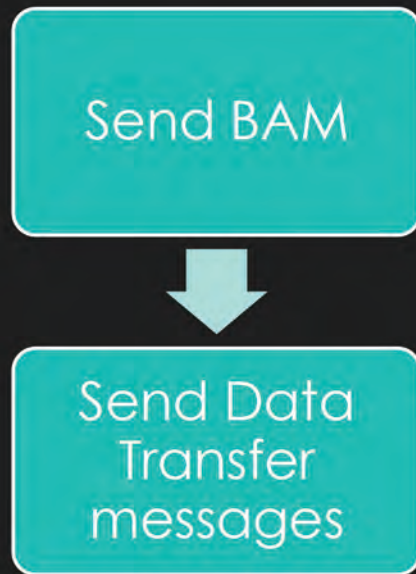
Packet 3: 0x030F101112FFFFFF

Data repacketized (PGN 0xFECA):

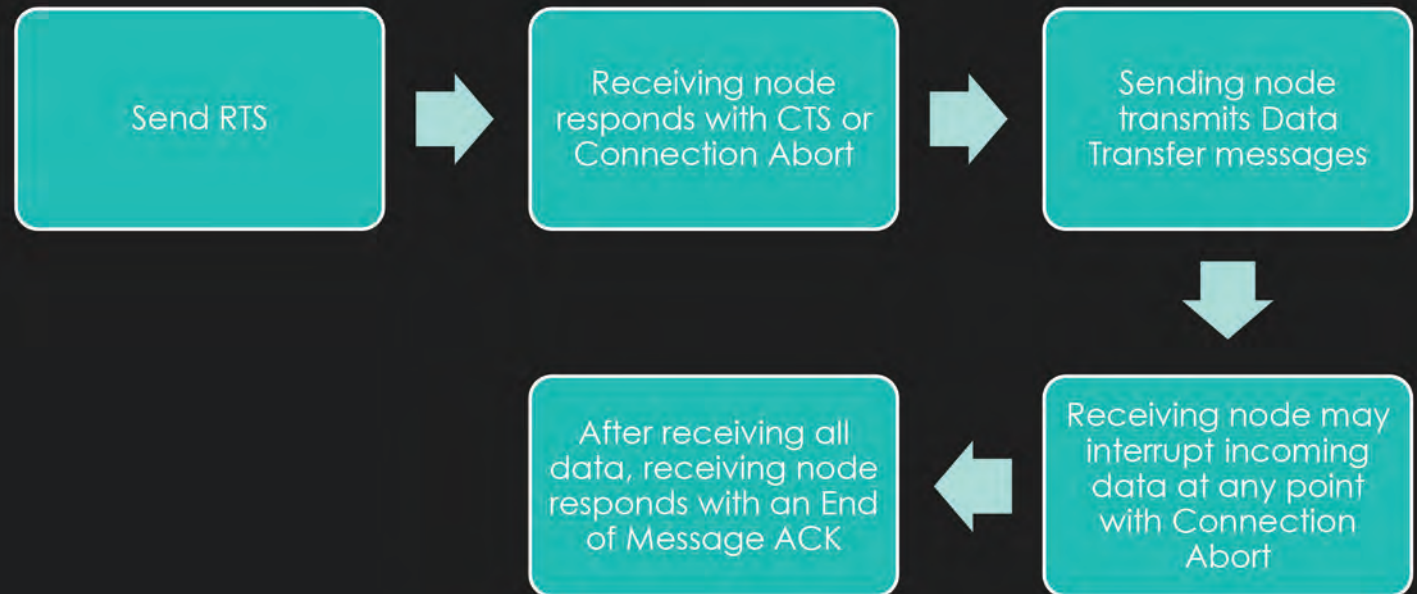
0x0102030405060708090A0B0C0D0E0F101112

Multi-packet messages

Broadcast Messages



Destination Specific Message



Multi-packet Destination Specific - RTS

- Transport Protocol – Connection Mgmt

PDU Format: 0xEC

PDU Specific: 0x0B (destination address)

Parameter Group: 0xEC00

Priority: 6

Source Address: 0x00

- CAN ID:

110 00 11101100 00001011 00000000

(0x18EC0B00)

- Data field:

Byte 1: 0x10

Byte 2-3: message size, in bytes

Byte 4: number of packets needed to send entire multi-packet message

Byte 5: max number of packets that can be sent in response to one CTS (no limit if 0xFF)

Byte 6-8: PGN of the multi-packet message

- Example RTS:

PGN 0x0700, 18 bytes long, sent over 3 packets

0x 10 1200 03 FF 000700

Multi-packet Destination Specific - CTS

- Transport Protocol – Connection Mgmt

PDU Format: 0xEC

PDU Specific: 0x00 (destination address)

Parameter Group: 0xEC00

Priority: 6

Source Address: 0x0B

- CAN ID:

110 00 11101100 00000000 00001011

(0x18EC000B)

- Data field:

Byte 1: 0x11

Byte 2: number of packets that receiving node will accept (can't exceed max from RTS)

Byte 3: next packet number to be sent

Byte 4-5: 0xFFFF

Byte 6-8: PGN of the multi-packet message

- Example CTS in response:

PGN 0x0700, start with packet 1

0x 11 03 01 FFFF 000700

Multi-packet Destination Specific – Data Transfer

- Transport Protocol – Data Transfer

PDU Format: 0xEB

PDU Specific: 0x0B (destination address)

Parameter Group: 0xEB00

Priority: 6

Source Address: 0x00

- CAN ID:

110 00 11101011 00001011 00000000

(0x18EB0B00)

- Data field:

Byte 1: 0x01-0xFF (sequence number)

Bytes 2-8: the data

- Example Data Transfer:

Packet 1: 0x0101020304050607

Packet 2: 0x0208090A0B0C0D0E

Packet 3: 0x030F101112FFFFFF

Data repacketized (PGN 0x0700):

0x0102030405060708090A0B0C0D0E0F101112

Multi-packet Destination Specific – Connection Abort

○ Transport Protocol – Connection Mgmt

PDU Format: 0xEC

PDU Specific: 0x00 (destination address)

Parameter Group: 0xEC00

Priority: 6

Source Address: 0x0B

○ CAN ID:

110 00 11101100 00000000 00001011

(0x18EC000B)

○ Data field:

Byte 1: 0xFF

Byte 2:

0x01 : node is already in a session

0x02 : node is lacking necessary resources

0x03 : timeout occurred

0x04-0xFF : reserved

Byte 3-5: 0FFFFFFF

Byte 6-8: PGN of the multi-packet message

○ Example:

PGN 0x0700, connection abort due to timeout

0x FF 03 FFFFFFFF 000700

Multi-packet Destination Specific – End of Message Acknowledgement

- Transport Protocol – Connection Mgmt

PDU Format: 0xEC

PDU Specific: 0x00 (destination address)

Parameter Group: 0xEC00

Priority: 6

Source Address: 0x0B

- CAN ID:

110 00 11101100 00000000 00001011

(0x18EC000B)

- Data field:

Byte 1: 0x13

Byte 2-3: message size, in bytes

Byte 4: total number of packets

Byte 5: 0xFF

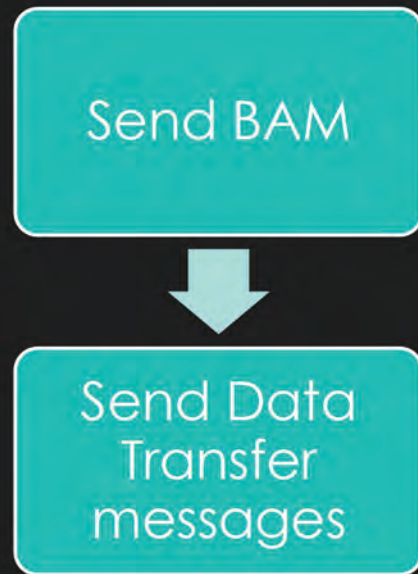
Byte 6-8: PGN of the multi-packet message

- Example:

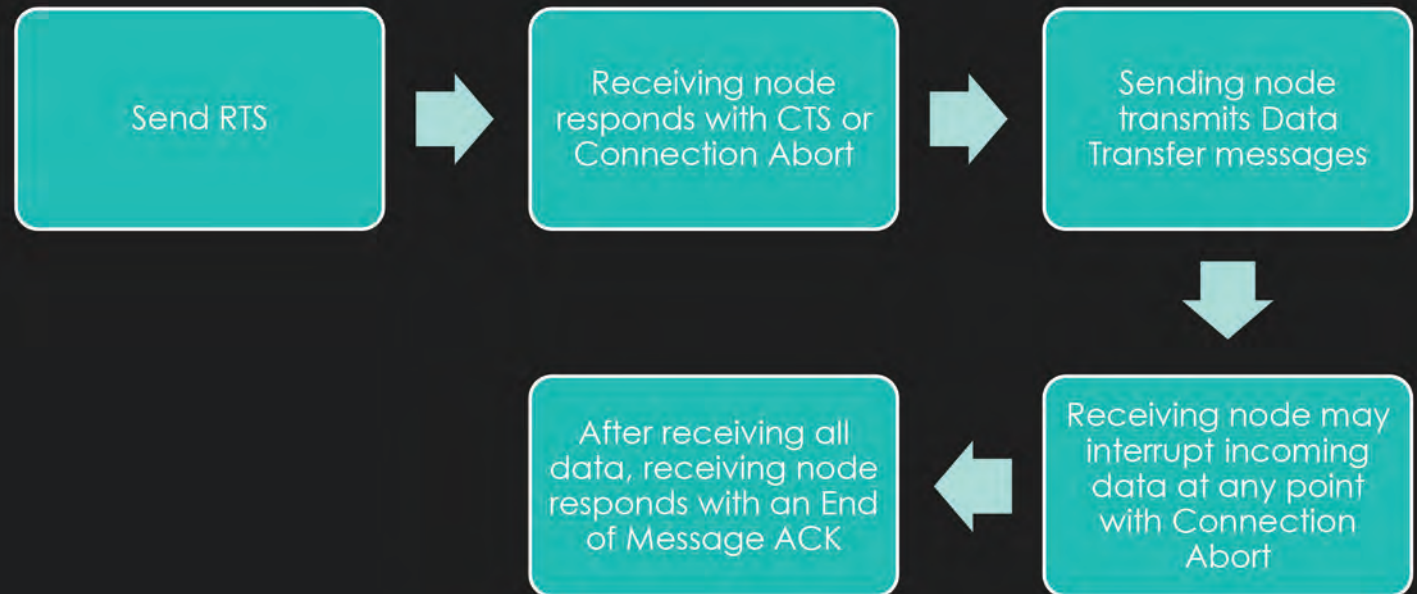
0x 13 1200 03 FF 000700

Multi-packet messages

Broadcast Messages



Destination Specific Message





Demo 5: reading multi- packet messages



Demo 6: sending multi- packet messages

Diagnostic Messages

- J1939-73 contains list of messages
- DM1: Read Diagnostic Trouble Codes (DTCs)
PGN: 0xFECA
- DM11: Clear all DTCs
PGN: 0xFED3
- DM13: Stop or start broadcast messages
PGN: 0xDF00
- DM14: Memory Access Request
PGN: 0xD900
- DM15: Memory Access Response
PGN: 0xD800
- DM16: Binary Data Transfer (READ)
PGN: 0xD700
- DM17: Boot Load Data (WRITE)
PGN: 0xD600
- DM18: Data Security
PGN: 0xD400

Unified Diagnostic Service - PGN

○ UDS

PDU Format: 0xDA

PDU Specific: 0x0B (destination address)

Parameter Group: 0xDA00

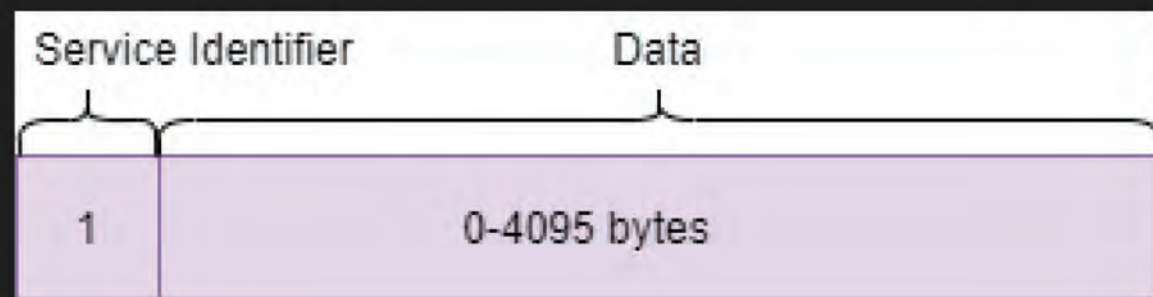
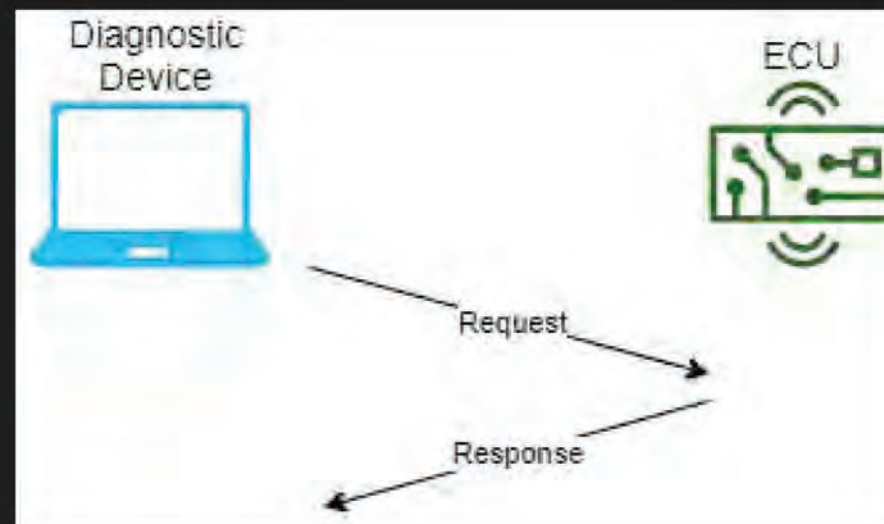
Priority: 6

Source Address: 0xF9

○ CAN ID:

110 00 11011010 00001011 11111001

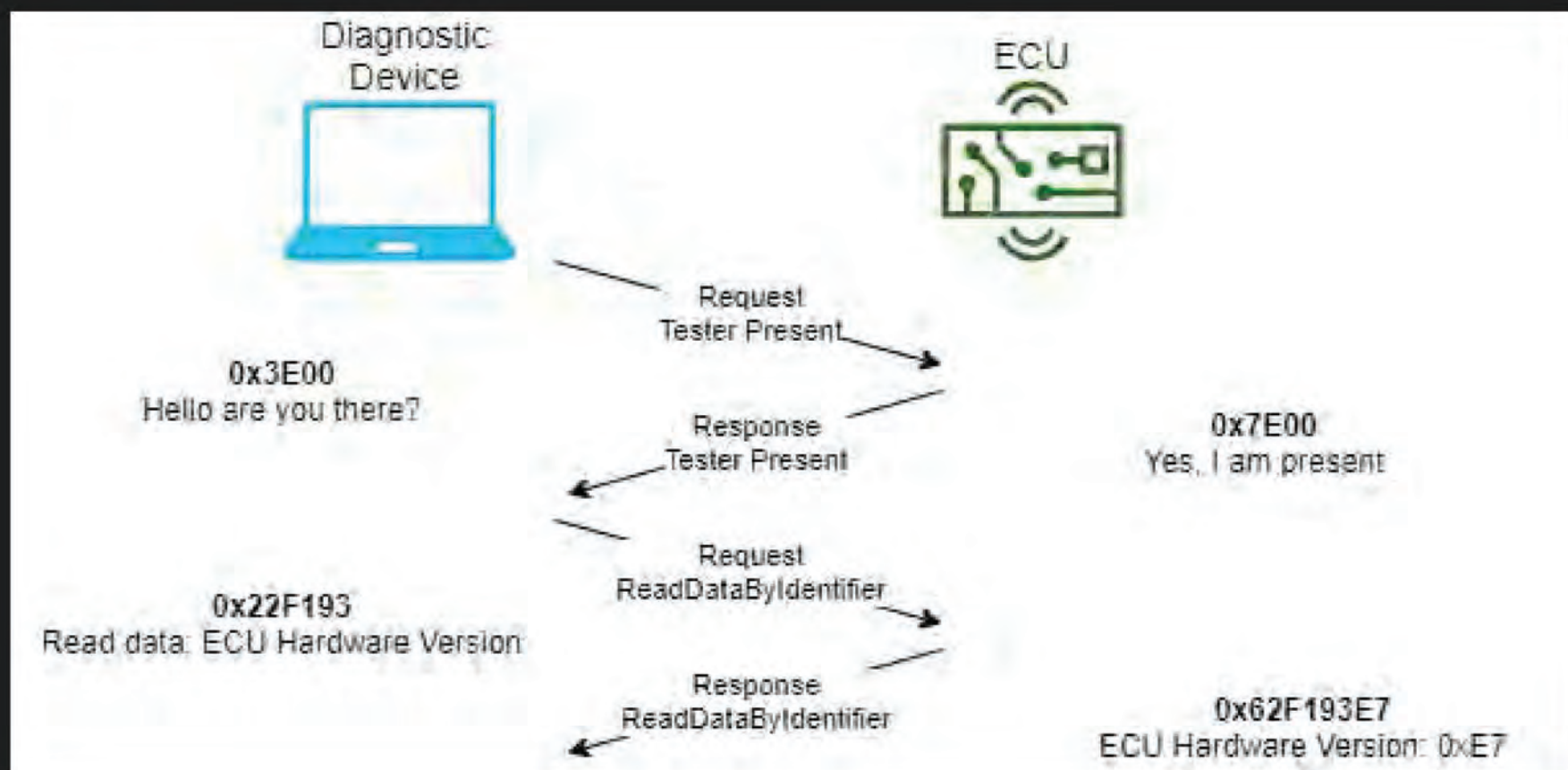
(0x18DA0BF9)



Unified Diagnostic Service - SID

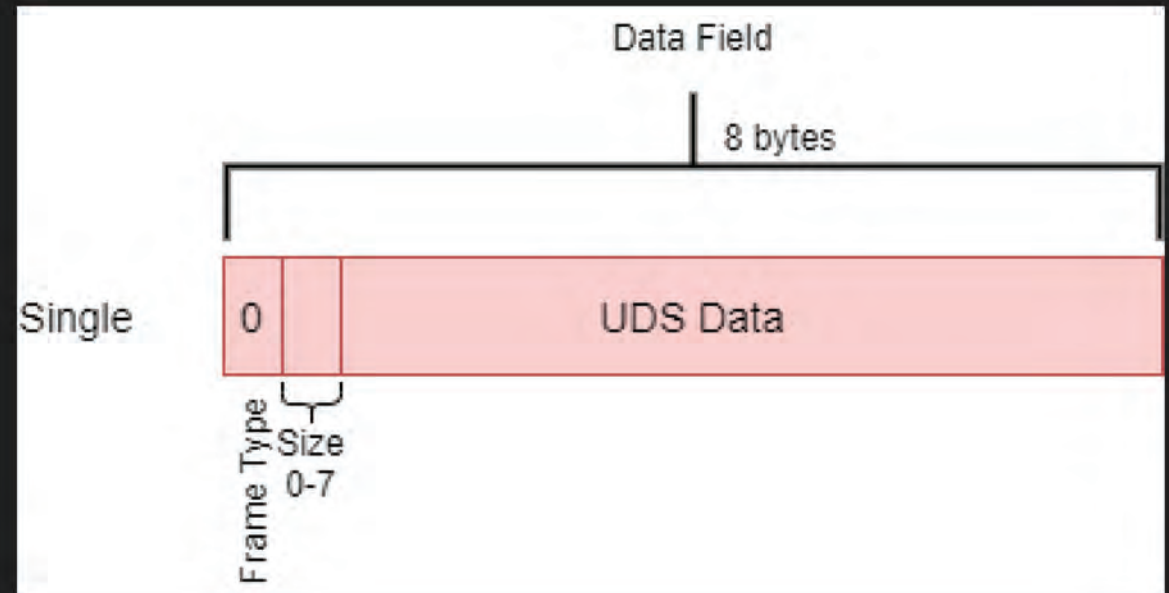
Service	Request SID	Response SID
Tester Present	0x3E	0x7E
Diagnostic Session Control	0x10	0x50
ECU Reset	0x11	0x51
Security Access	0x27	0x67
Read/Write Data By Identifier	0x22 / 0x2E	0x62 / 0x6E
Read/Write Memory By Address	0x23 / 0x3D	0x63 / 0x7D
Read/Clear DTC Information	0x19 / 0x14	0x59 / 0x54
Negative Response		0x7F

Unified Diagnostic Service – Example Flow

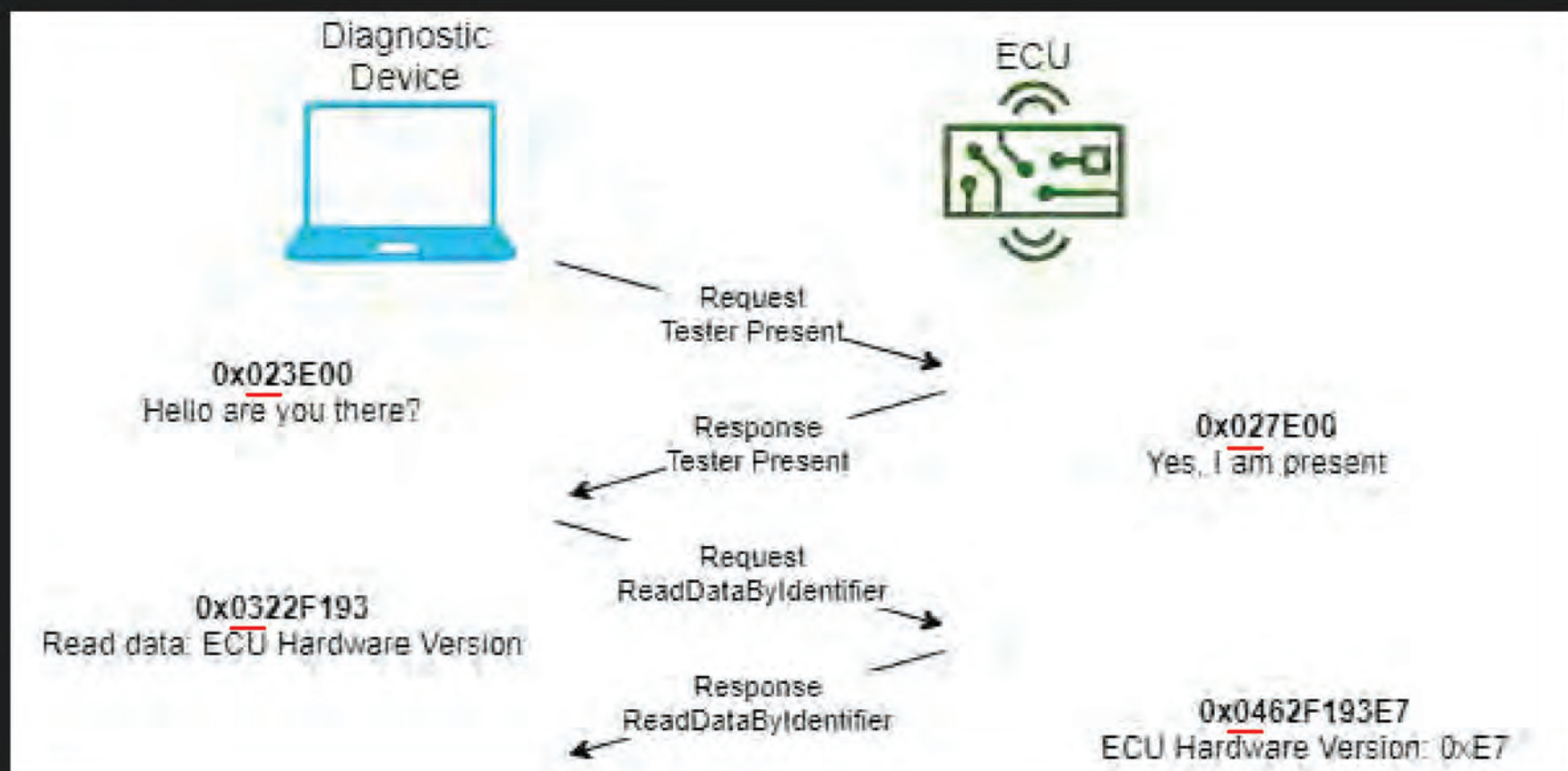


Unified Diagnostic Service – ISO-TP

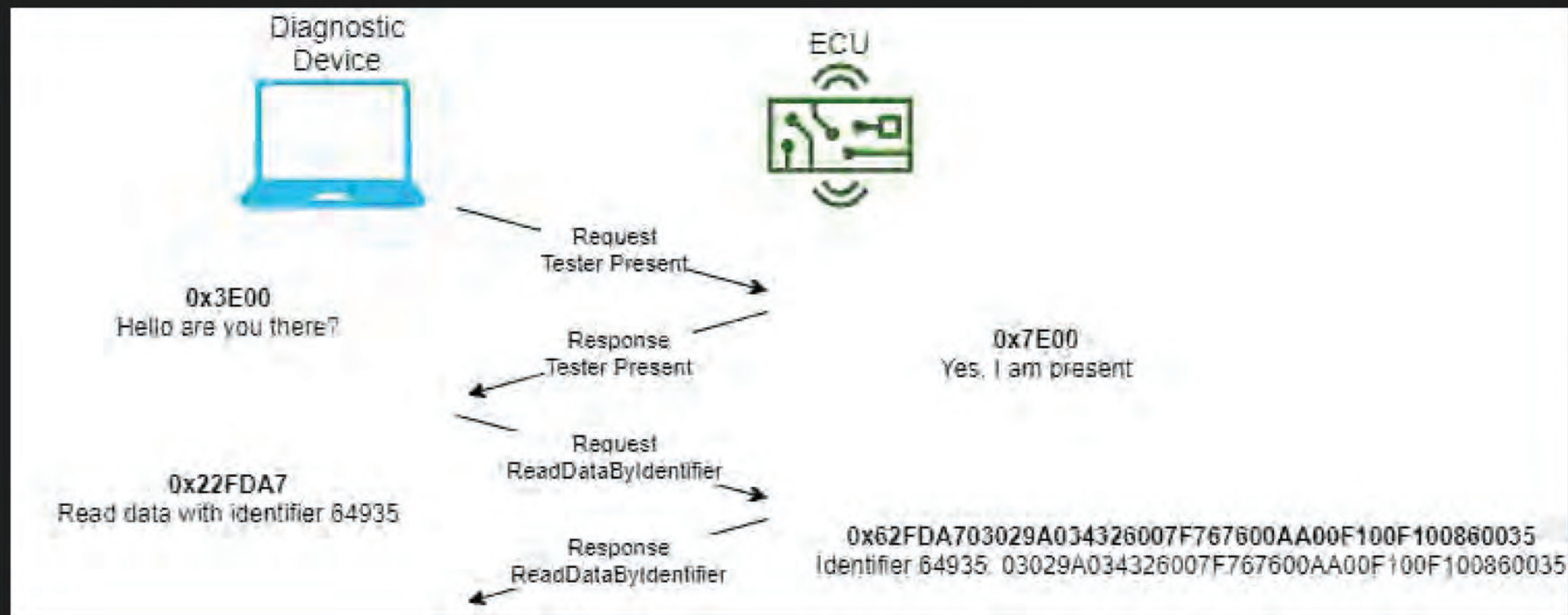
- ISO 15765-2 (ISO-TP) in use for sending UDS packets rather than the standard J1939 Transport Protocol
- Packets 0-7 bytes long
Single frame (frame type 0)
- Example
UDS tester present message: 0x3E00
Data field: 0x023E00



Unified Diagnostic Service – ISO-TP



Unified Diagnostic Service – ISO-TP



Unified Diagnostic Service – ISO-TP

- Packets 8-4085 bytes long

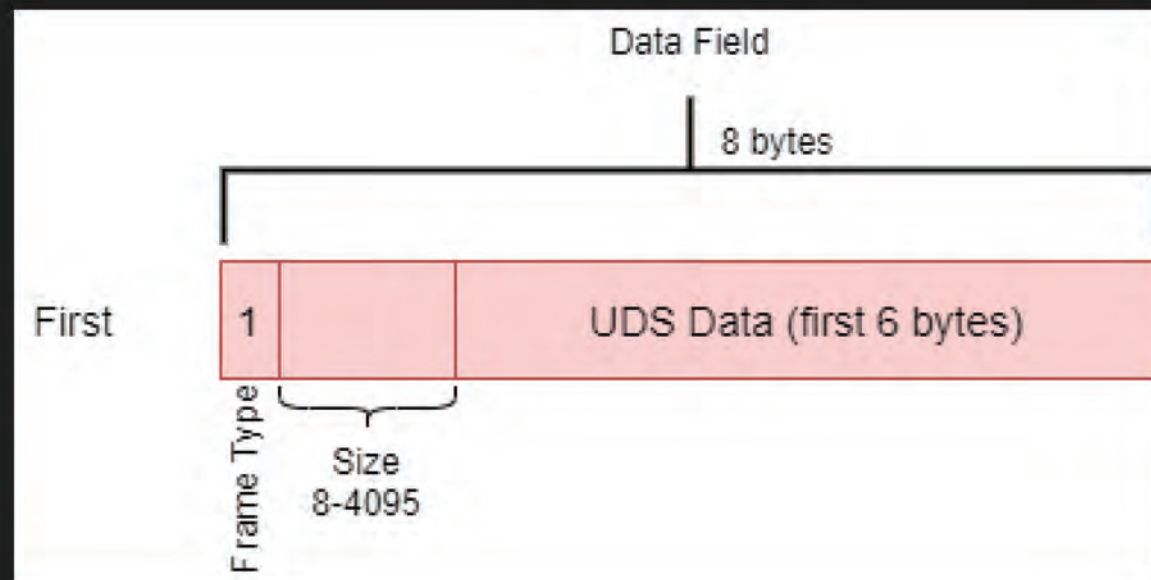
1. Sender sends First Frame (type 1)
2. Receiver sends Flow Frame (type 3)
3. Sender sends Consecutive Frames (type 2)

- Example

UDS ReadDataByIdentifier response:

0x62FDA703029A034326007F767600AA00F100F100860035

First Frame Data field: 0x101762FDA703029A



Unified Diagnostic Service – ISO-TP

- Packets 8-4085 bytes long
 1. Sender sends First Frame (type 1)
 2. Receiver sends Flow Frame (type 3)
 3. Sender sends Consecutive Frames (type 2)

- FC Flag:

0: Clear to send

1: Wait

2: Overflow/abort

- Block Size:

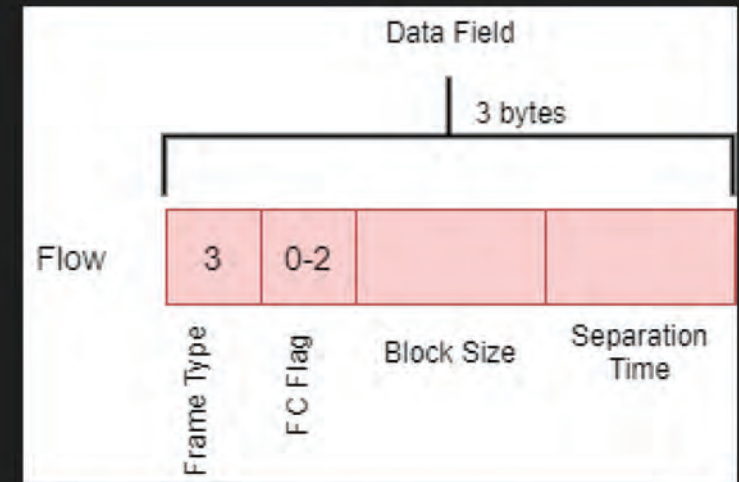
0: no flow control

> 0: # frames to
send between flow
control frame

- ST:

0-127: milliseconds
between frames

241-249: 100-900
microseconds



- Example Data field:

CTS, no flow control, 50 ms between

0x300032

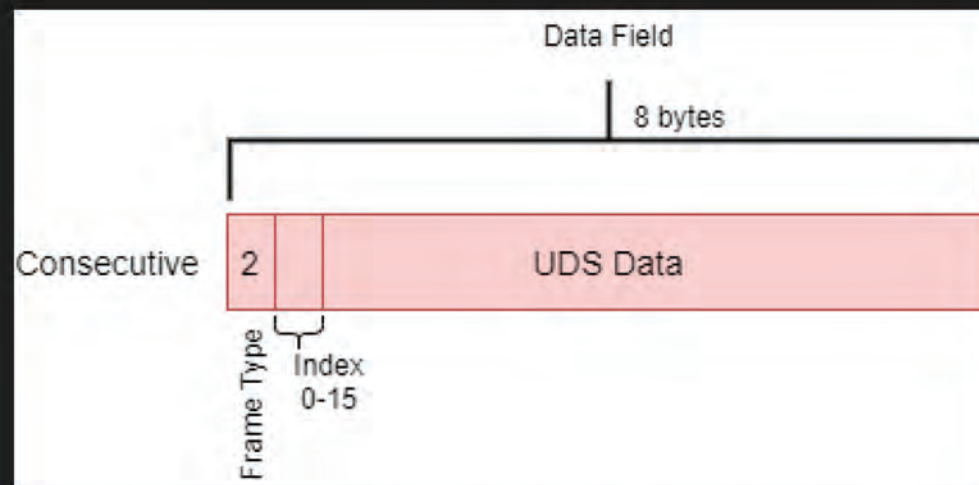
Unified Diagnostic Service – ISO-TP

- Packets 8-4085 bytes long
 1. Sender sends First Frame (type 1)
 2. Receiver sends Flow Frame (type 3)
 3. Sender sends **Consecutive** Frames (type 2)

- Example

UDS ReadDataByIdentifier response:

0x62FDA703029A034326007F767600AA00F100F100860035



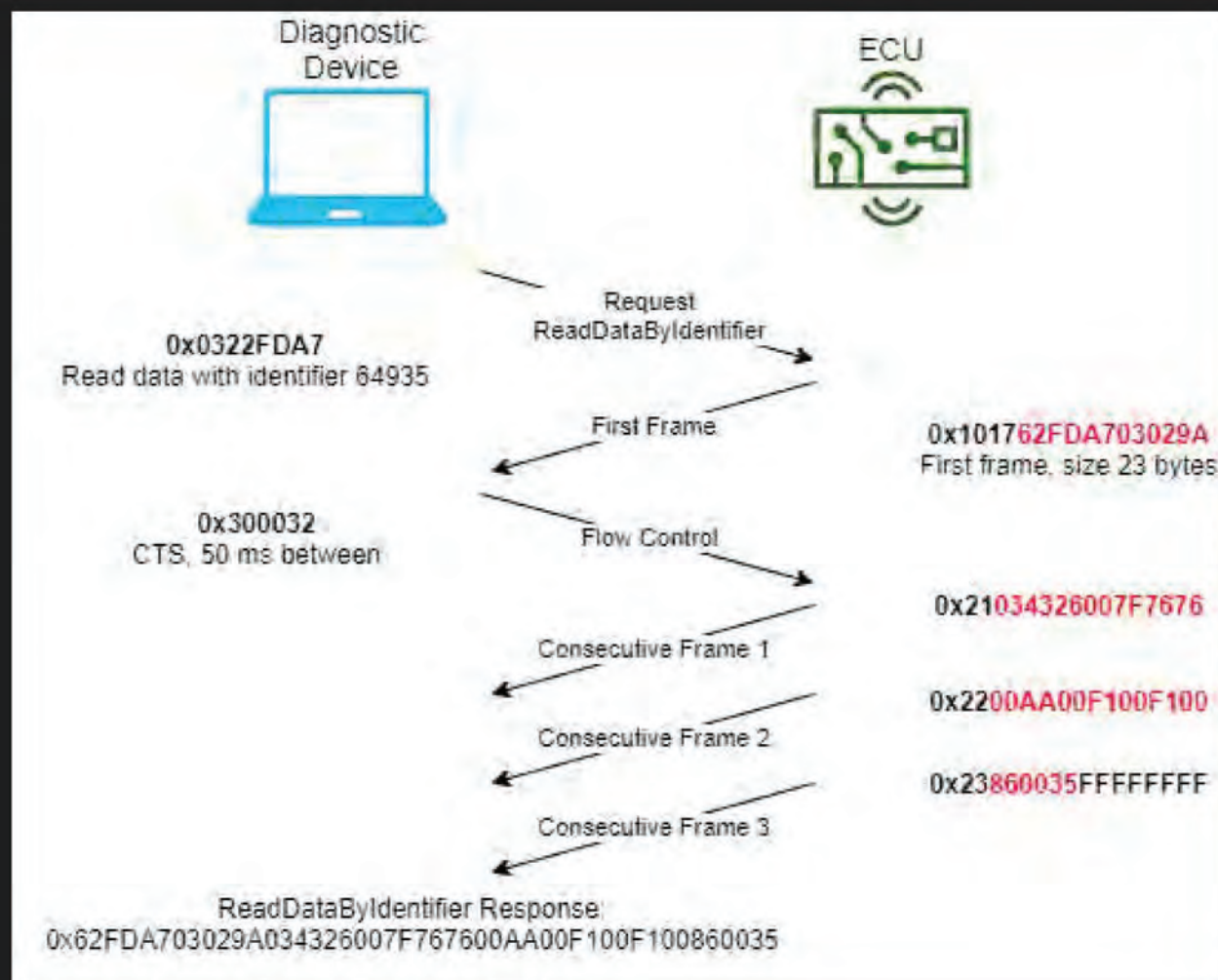
First Frame Data field: 0x101762FDA703029A

Consecutive Frame 1: 0x21034326007F7676

Consecutive Frame 2: 0x2200AA00F100F100

Consecutive Frame 3: 0x23860035FFFFFFFF

Unified Diagnostic Service – ISO-TP



Truck Networks

CAN (Controller Area Network)
- Physical and Data Link layers

J1939
- Higher layer protocol based on CAN
- Newer & faster

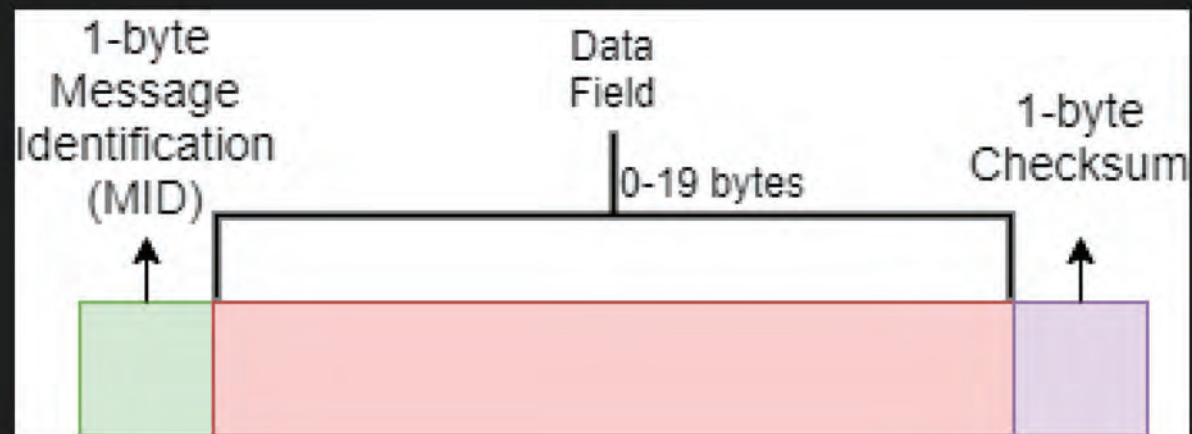
J1708
- Physical layer
- Primarily on older vehicles

J1587
- Higher layer protocol that runs on J1708

J2497
(PLC4TRUCKS)
- Basically J1708 but on trailer power lines

J1708

- Physical layer based on RS-485 bus, except it doesn't use the termination resistors
- Baud rate: 9600 bit/s
- Messages are up to 21 bytes long



Truck Networks

CAN (Controller Area Network)
- Physical and Data Link layers

J1939
- Higher layer protocol based on CAN
- Newer & faster

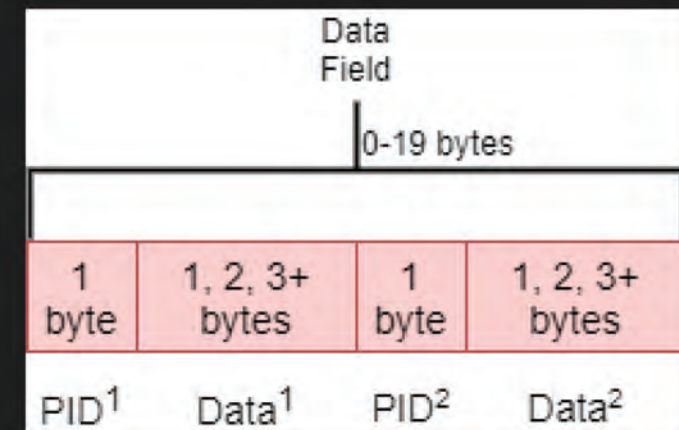
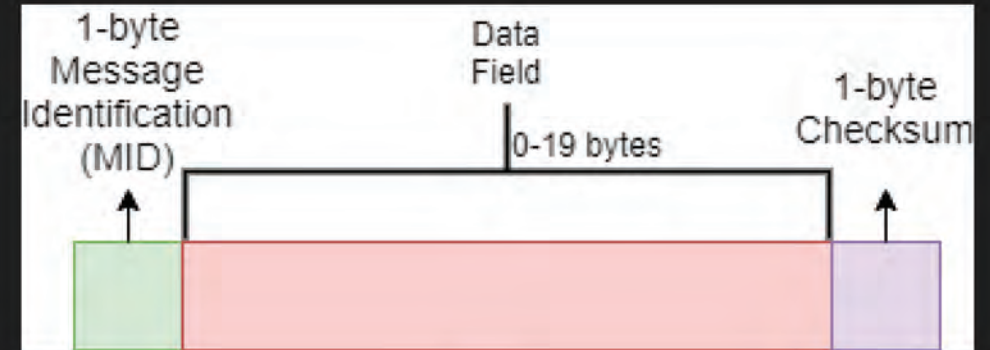
J1708
- Physical layer
- Primarily on older vehicles

J1587
- Higher layer protocol that runs on J1708

J2497
(PLC4TRUCKS)
- Basically J1708 but on trailer power lines

J1587

- Transport and application layer that runs on top of J1708
- Data exchange, diagnostics, navigation, etc.
- Has a Connection Oriented Transport Service (COTS) for sending messages >21 bytes long

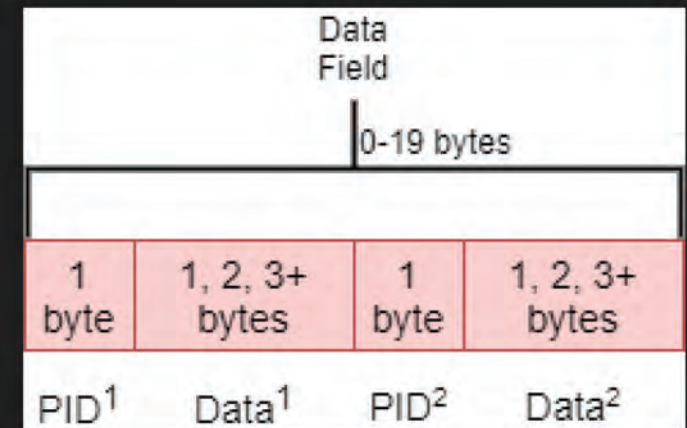


J1587 - MID

MID	Description
0-127	Reserved
128	Engine #1
129	Turbocharger
130	Transmission
131	Power Takeoff
132-135	Axle related
136-139	Brake related
140	Instrument Cluster
...	...
154	Diagnostic Systems

J1587 - PID

PIDs	Data Length	Example/ Description
0-127	1 byte	0 = Request 8 = Brake pressure low
128-191	2 bytes	132 = Mass air flow 134 = Wheel speed sensor
192-253	3+ bytes	194-196 = Diagnostics 225-227 = Dashboard text display
254	Variable	Manufacturer Specific
255	Variable	PID extension



Truck Networks

CAN (Controller Area Network)
- Physical and Data Link layers

J1939
- Higher layer protocol based on CAN
- Newer & faster

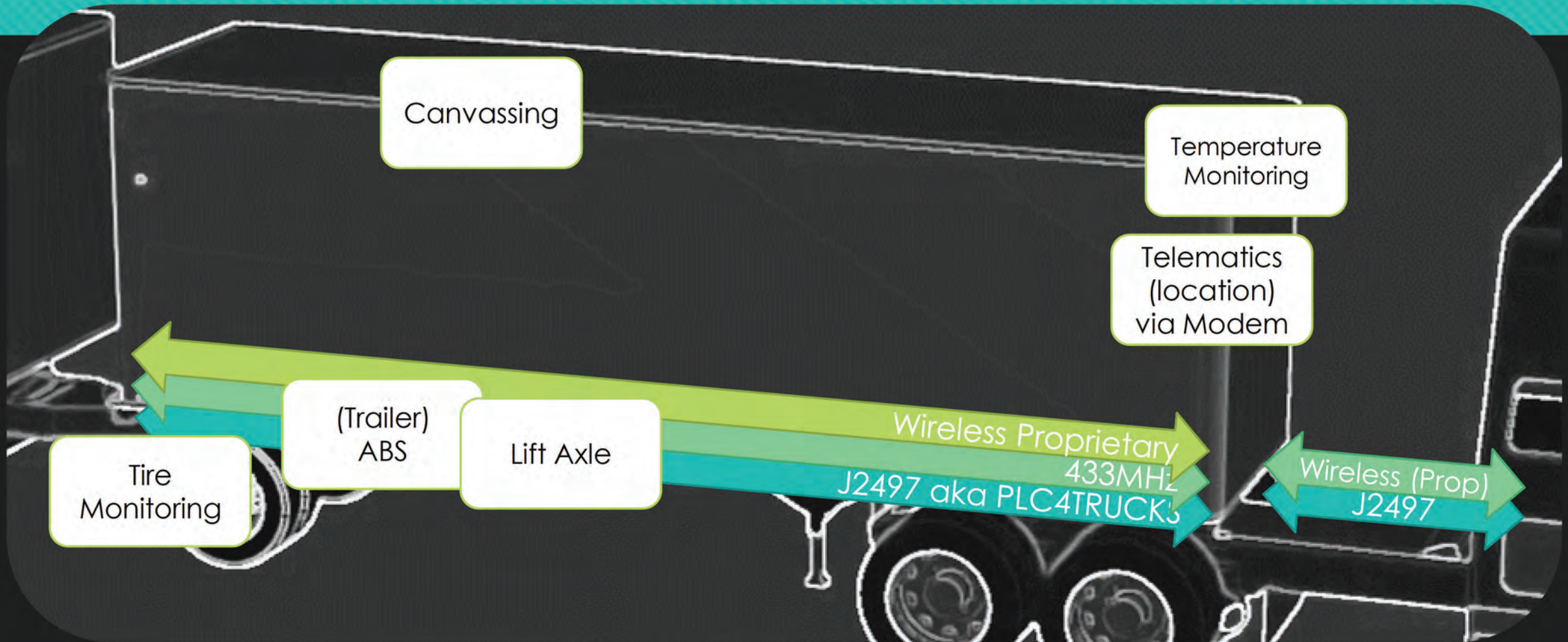
J1708
- Physical layer
- Primarily on older vehicles

J1587
- Higher layer protocol that runs on J1708

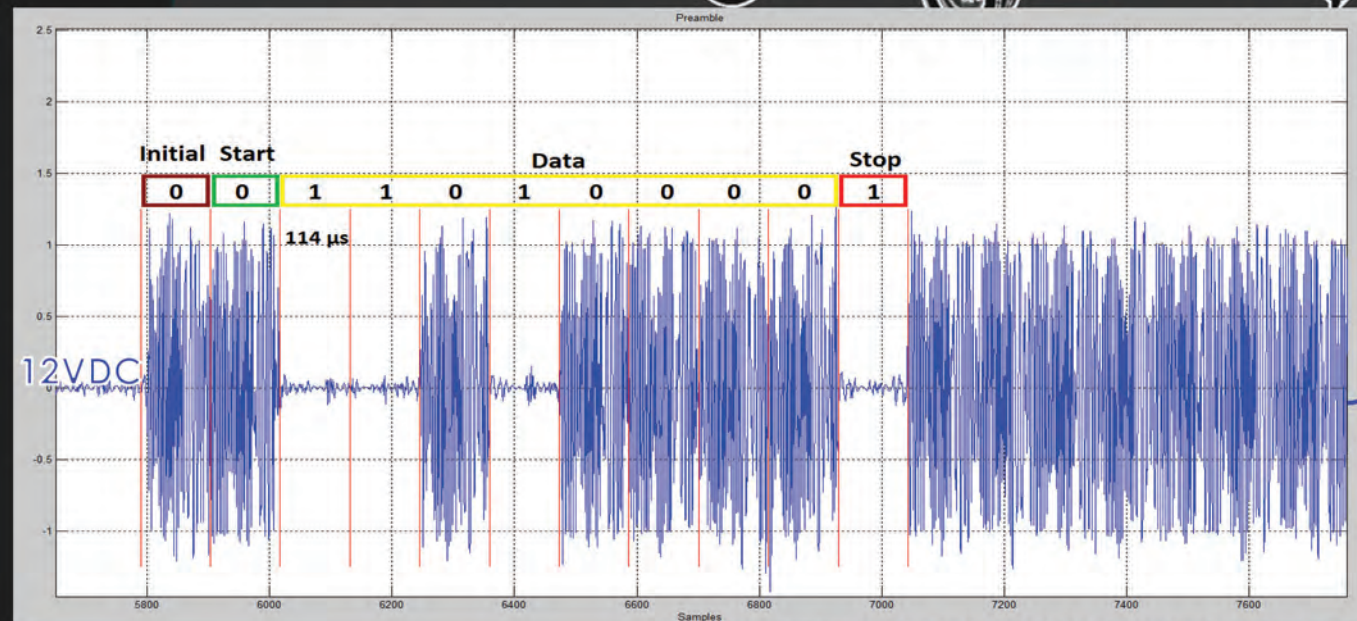
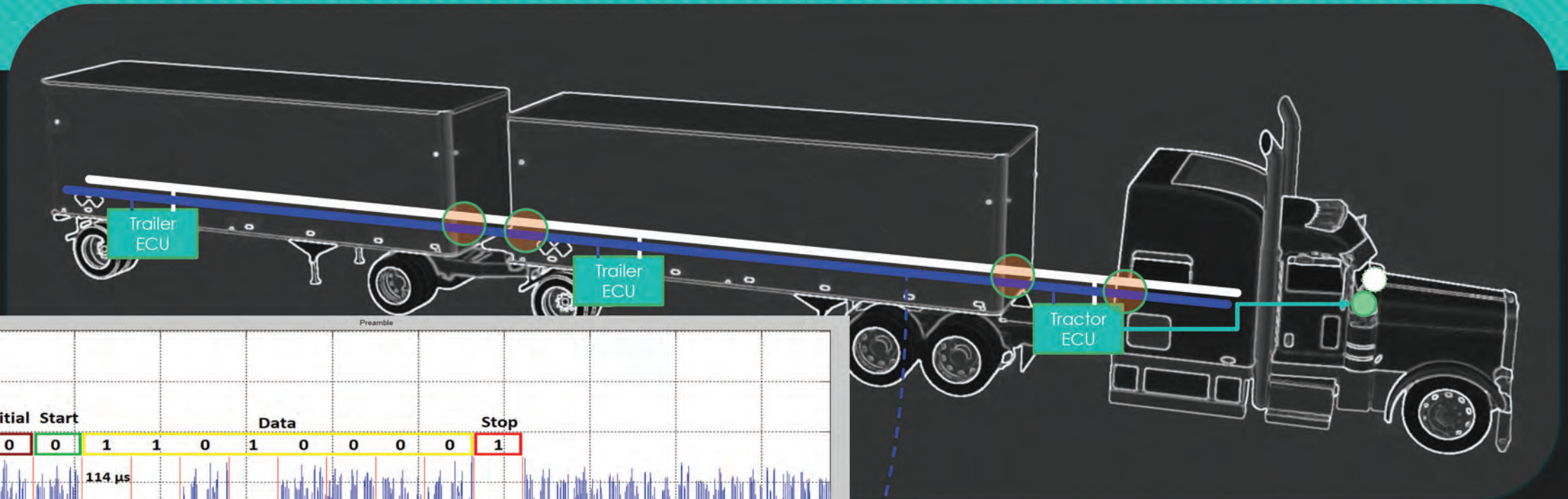
J2497
(PLC4TRUCKS)
- Basically J1708 but on trailer power lines

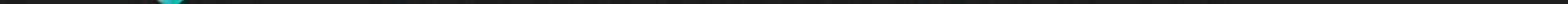
About Trucking: Trailers

- The other things that roll
- These outnumber tractors (in North America)
- Many features today



Power Line Carrier (for trucks)



- 



Common PLC Messages (on Trailers)

- In our testing the messages we've seen are limited to:
 - ABS Fault Lamp on and off
 - Diagnostics
- 2005 FMCSA-PSV-06-001 report suggests that there should also be:
 - Axle weigh systems
 - Yaw sensors
 - Door latch sensors

Finding J2497 (1/5)

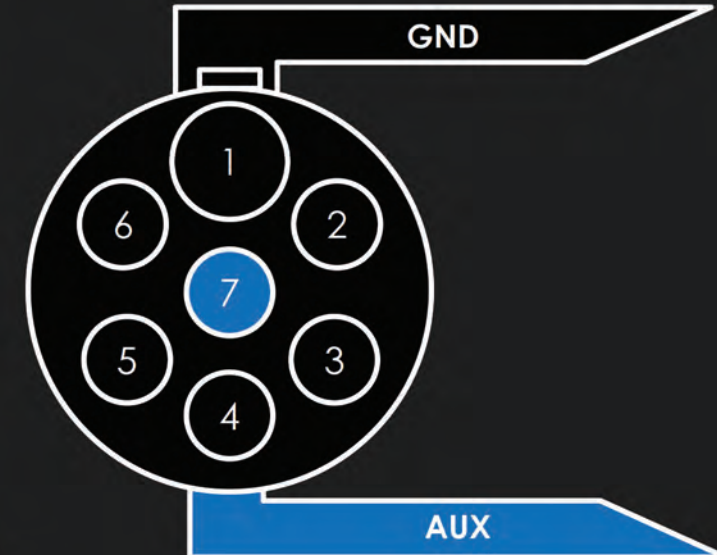
- Will always be on the power pin (AUX) of the trailer J560 connector ➡ (at back of tractor / front of trailer)



CC BY-SA MobiusDaXter

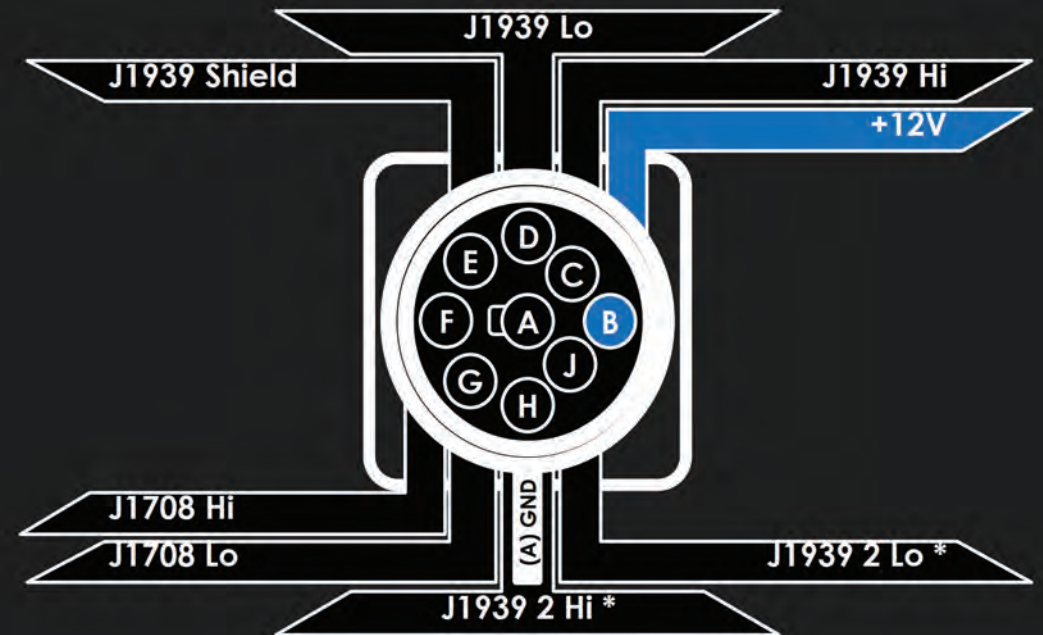
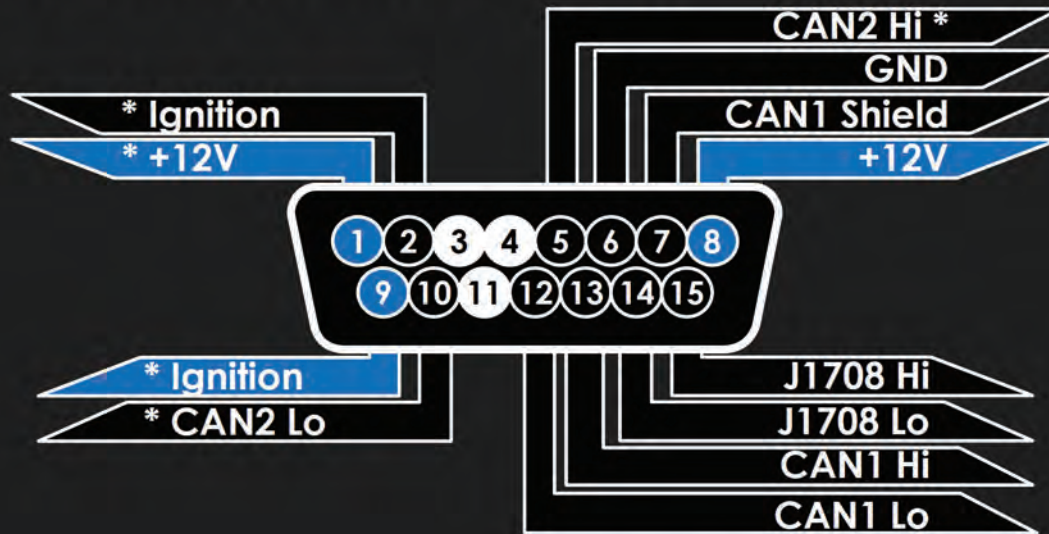


www.ebay.ca/itm/Bendix-ABS-Trailer-Remote-Diagnostic-Unit-TRDU-PLC-Adapter-9-pin-Connection



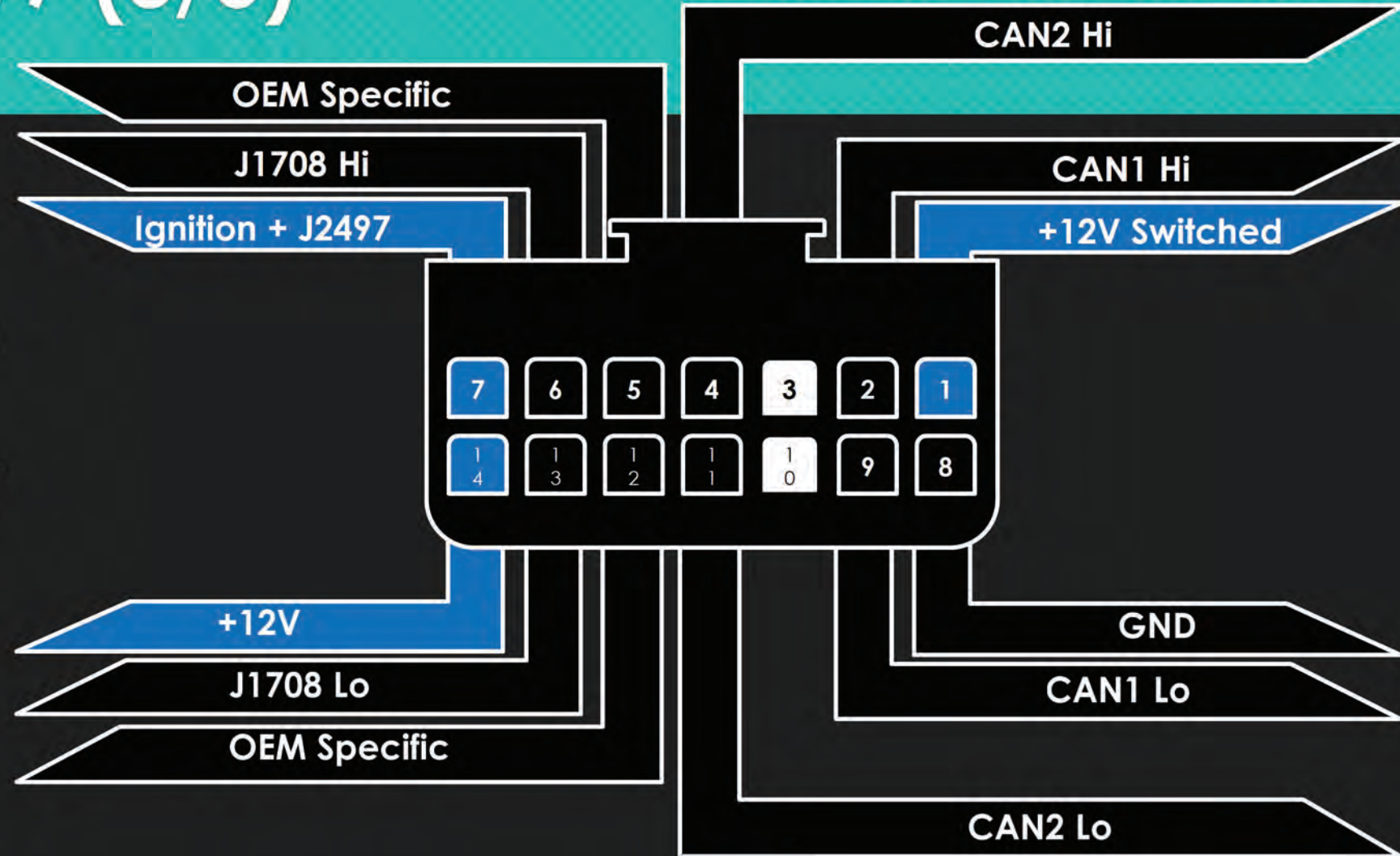
Finding J2497 (2/5)

- Might be on the power pins of the diagnostics connector
- What you find could be filtered/segmented from the trailer.



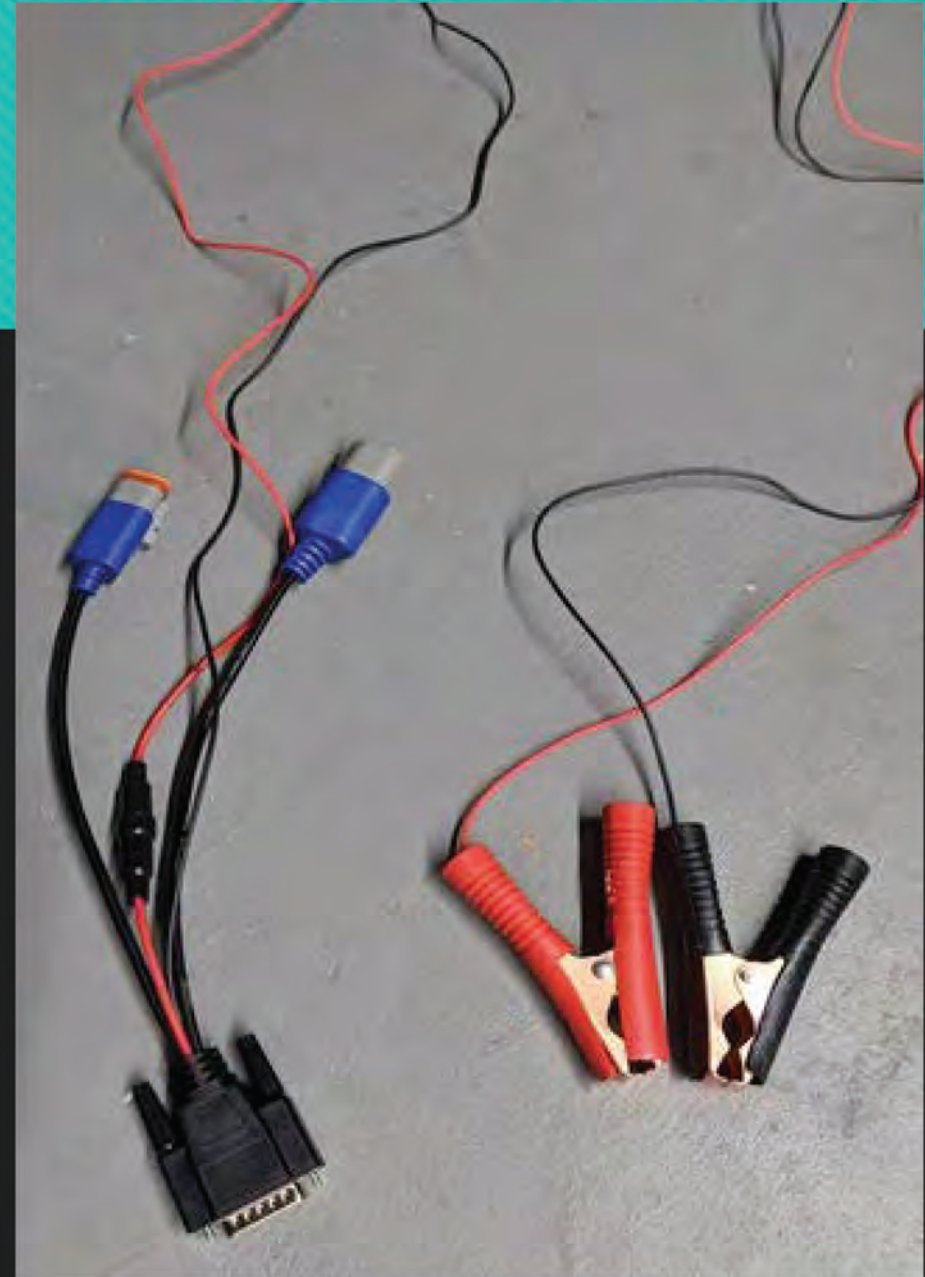
Finding J2497 (3/5)

- Should be on the RP1226 (Aftermarket/Telematics) Connector



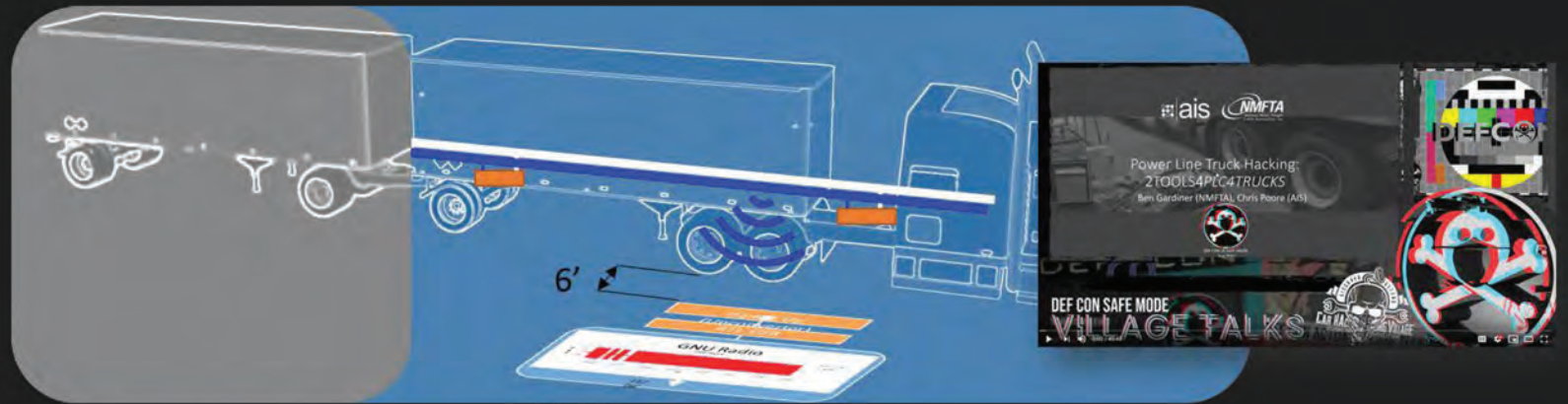
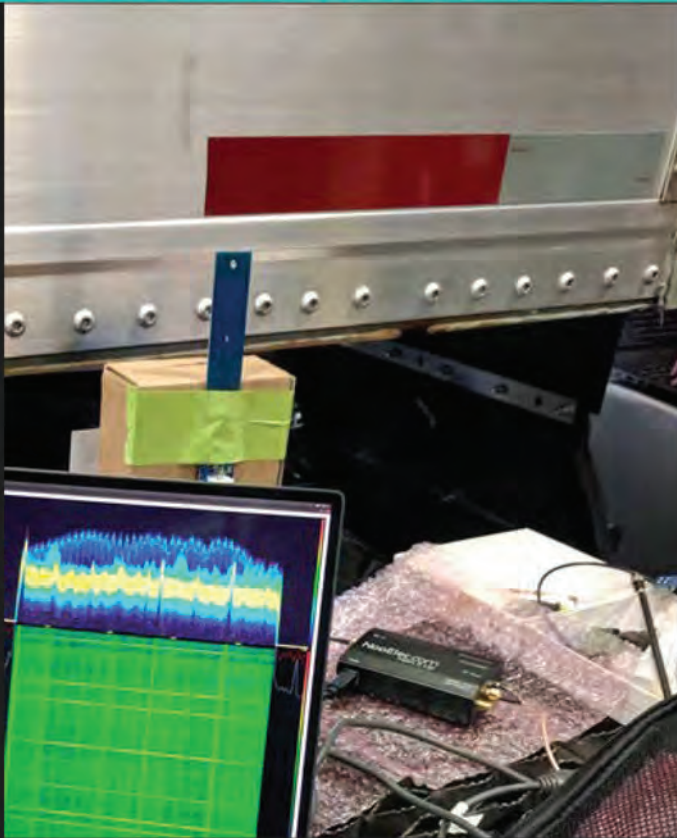
Finding J2497 (4/5)

- Might be on the battery terminals -- but what you find could be filtered/segmented from the trailer.



Finding J2497 (5/5)

- Might just radiate away from the trailer. [ICSA-20-219-01](#)



Poore, Chris & Gardiner Ben.

[Power Line Truck Hacking: 2TOOLS4PLC4TRUCKS](#)

Interfacing with PLC

- Traditional Diagnostics Adapters
 - Convert PLC <-> J1708 adapter pins
- e.g. DG Technologies PLC TestCon 600USD
- e.g. Nexiq 604020 330USD
- Converts from PLC on power pins of DB15 to J1708 pins on another DB15
- These aren't cheap.
- Intellon P485 becoming harder to source
- These also can't do 'weird' things to PLC. They are limited by J1708 interface.



<https://www.dgtech.com/store/plc-testcon-with-battery-cable.html>



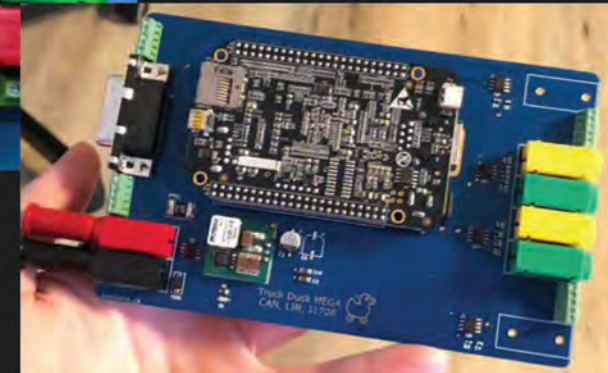
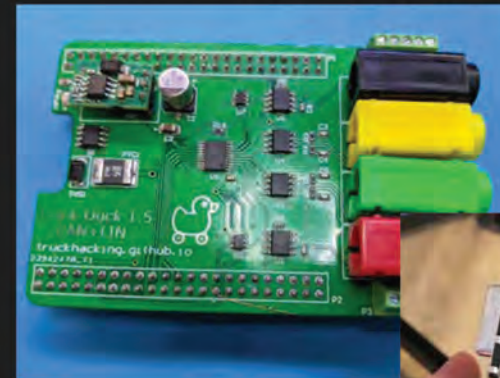
Nexiq PLC adapter (FindItParts.com)

Interfacing with J1708 (to interface with PLC)

- An RP1210 Adapter with J1708 support (most)
 - Needed for manufacturer diagnostics tools
 - E.g. DG Tech DPA4, DPA5, Nexiq USBLINK
- Truck Duck beaglebone cape(s)
 - by [sixvolt](#) & [haystack](#), @DEF CON 24
 - Later revisions 1.5 YEET and MEGA
 - [py-hv-networks](#) for J1708 (and SocketCAN for J1939)



truckhacking.github.io



Tools for Hacking Truck Networks

- Pretty_j1939.py
for decoding J1939
- TruckDuck/TruckCape
reading/sending J1939
- CanCat
CAN swiss-army knife
- Vector CANoe
\$\$\$ commercial tool for reading / ECU development
- Vehicle Spy
\$ another commercial tool, not as expensive although doesn't have as many features

Challenge 2 (with any remaining time)

- Connect to ECU with diagnostic software and view any UDS or proprietary messages
- Can you think of anything malicious you may want to attempt?

Thank you!
Questions?