COLORADO STATE UNIVERSITY

CyberBoat Challenge - Introduction to CAN, J1939 and NMEA2000

Jeremy Daily, Ph.D., P.E.
Associate Professor
jeremy.daily@colostate.edu

SYSTEMS ENGINEERING
COLORADO STATE UNIVERSITY

Agenda

- Personal introduction with some boating projects
- Physical Connections for the Controller Area Network (CAN)
- Reading CAN messages
- Writing CAN Messages
- Interpreting J1939 Messages
  - Transport Protocol for Long Messages
- Interpreting NMEA 2000 Messages
  - Fast Packet for Long Messages
- Address Claiming
  - Denial of Service with an Address Claim Attack

Introducing Dr. Jeremy Daily

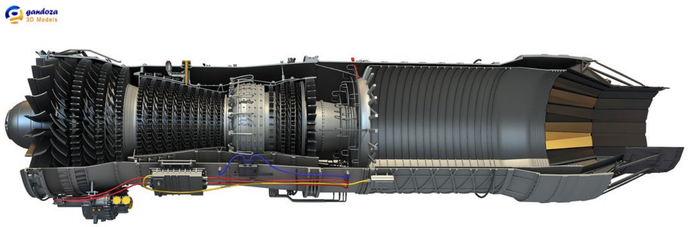
US Air Force Flightline Electronics Maintenance

Synercon Technologies

Startup Company for Heavy Vehicle Digital Forensics

WRIGHT STATE UNIVERSITY

Formal Education in Mechanical Engineering



UNIVERSAL TECHNOLOGY COMPANY

Contracted Aerospace Engineer at Wright-Patterson AFB

CYBERTRUCK CHALLENGE®

Co-Founder and Director

THE UNIVERSITY of TULSA

Faculty in the Department of Mechanical Engineering

SYSTEMS ENGINEERING

COLORADO STATE UNIVERSITY

Sailboat

1977 American Mariner

Leeks Marina

Jackson Lake

Grand Teton National Park, WY

Ski Boat

1999 Sport Nautique
by
Correct Craft

South Bay of Horsetooth Reservoir
Fort Collins, CO

Fishing Autopilot with Arduino and Digital Compass

Surf Boat

2012 Mastercraft X-30

South Bay of Horsetooth Reservoir
Fort Collins, CO

(this one has CAN)

Skiing on Jackson Lake

Controlling the Engine

- Only 1 mechanical linkage from the helm to the engine compartment for shifting.
- How does the engine know what speed to run?

In-Vessel Networking

- How do the instruments know what values to display?
- What speed should the engine run?
- How do we efficiently share information?

Most modern boats use NMEA 2000, which is built on SAE J1939, which is built on a Controller Area Network (CAN)

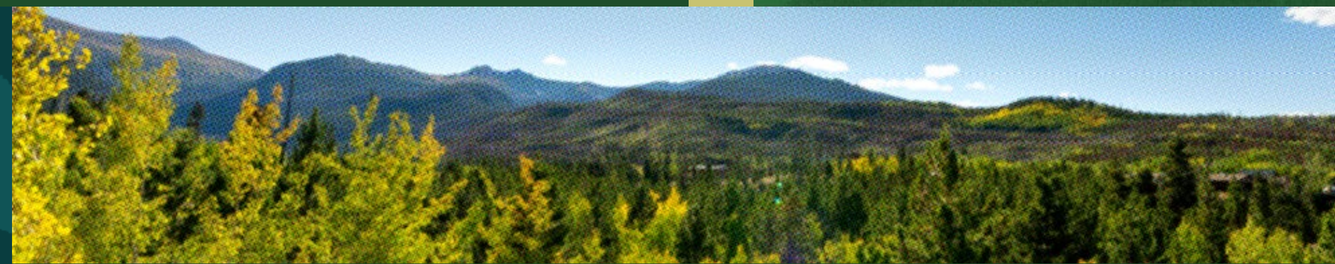
Connecting to CAN on the Mastercraft

Personal Goals:

- Learn the modules
- Record data
- Add supplemental systems
  - Ballast Tank Fill Indicators
  - Rudder Position
  - Upgraded Stereo

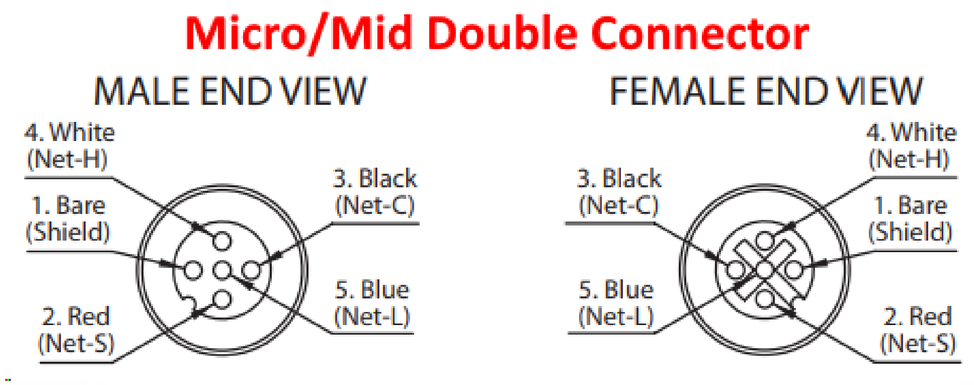
Physical Connections

Recognizing the connectors, wires, and signaling to make controller area network communication on boats a reality.



NMEA 2000 Connector

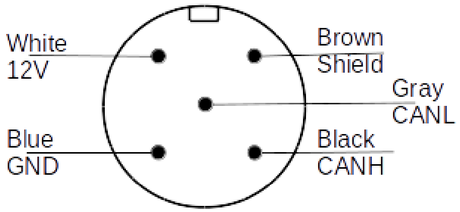
https://www.amazon.com/Elecbee-Female-Straight-Waterproof-Connector/dp/B09S5ZVGSK



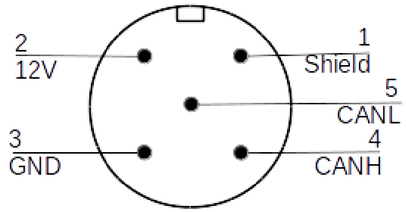
https://www.maretron.com/wp-content/phpkbv95/assets/img_5c828a06d6d94.png

https://www.amazon.com/dp/B09JC9YMC5/

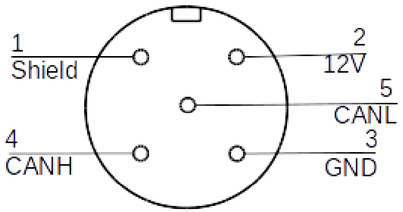
5 Pin M12 Connector, A Coded



Male



Male



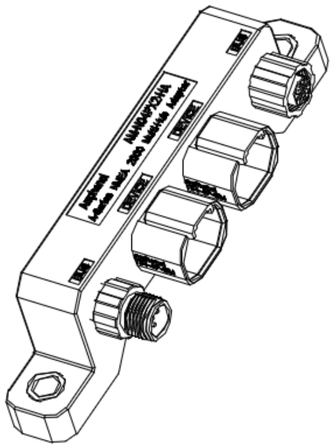
Female

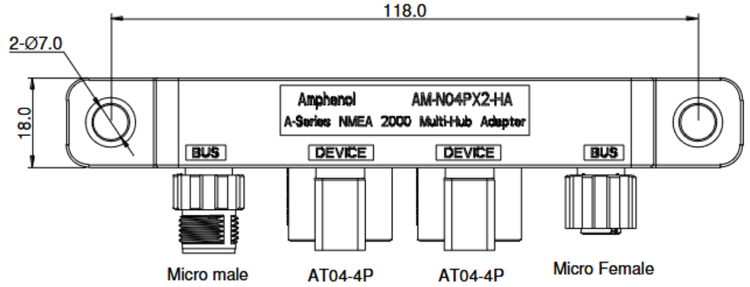
Pin	Name	Color vyacht	Color Garmin
1	Shield	Brown	Blank
2	12V / NET-S	White	Red
3	Ground / NET-C	Blue	Black
4	CAN-H / NET-Hi	Black	White
5	CAN-L / NET-Lo	Gray	Blue

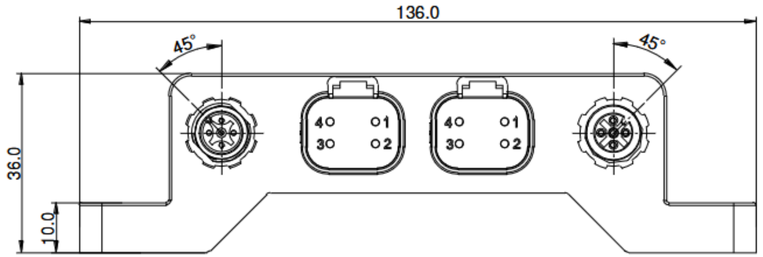
https://www.vyacht.net/img/NMEA2000-pinout.png

M12 to Deutsch 4-pin

REVISIONS					
REV	ECO	DESCRIPTION	DATE	BY	APPR
A1	-	FIRST RELEASE	Aug.11,2015	Tod	TOMMY
A2	-	Add The Angle For Micro Male And Female	Aug.18,2015	Drack	TOMMY







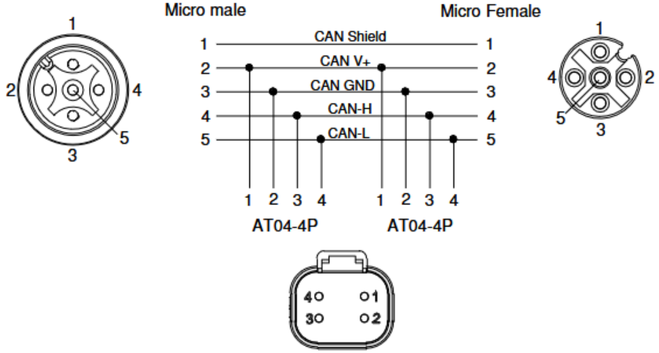
NOTES: UNLESS OTHERWISE SPECIFIED

1. MATERIAL:
   COUPLING NUT: PA66 UL94V-0
   INSULATION INSERT: PA66,UL94 V0
   SEAL: SILICON RUBBER
   OVERMOLD MATERIAL: THERMOPLASTIC COLOR: BLACK
   CONTACT: GOLD FLASH PLATING OVER COPPER ALLOY

2. SPECIFICATIONS:
   2.1 CURRENT RATING: 4 AMPS
   2.2 VOLTAGE RATING: 60V AC/DC
   2.3 OPERATING TEMPERATURE: -20°C TO +130°C
   2.4 DIELECTRIC WITHSTANDING VOLTAGE: LESS THAN 2 MILLIAMPS CURRENT LEAKAGE @ 1000 VOLTS AC.
   2.5 DEGREE OF PROTECTION: IP67 (MATED CONDITION)
   2.6 DEGREE OF POLLUTION: 3 PER UL840
   2.7 OVERVOLTAGE CATEGORY: III PER UL840
   2.8 MATING CYCLE DURABILITY: >100 CYCLES
   2.9 RoHS COMPLIANT

3. ALL DIMENSIONS ARE FOR REFERENCE USE ONLY.

Wire diagram



QUANTITY	PART NUMBER	DESCRIPTION	ITEM
	SEE PART NUMBER CHART		

MATERIALS LIST

UNLESS OTHERWISE SPECIFIED
1) All dimensions are in metric(mm).
2) Tolerances are as follows:
1 PL DEC ±0.30 | Fractions ±1/64
2 PL DEC ±0.15 | Angles ±1°
3 PL DEC ±0.08
3) Note reference – X

MATERIAL SPECIFICATIONS:

PROCESS SPECIFICATIONS:

NEXT ASSY:

SIGNATURES	DATE
DRAWN: Drack	Aug.18,2015
CHECKED:	
ENGINEER:	
APPROVAL:	
CUSTOMER:	

THIS DRAWING IS SUPPLIED FOR INFORMATION ONLY. DESIGN FEATURES, SPECIFICATIONS AND PERFORMANCE DATA SHOWN HEREON ARE THE PROPERTY OF THE AMPHENOL CORPORATION. NO RIGHTS OF REPRODUCTION ARE IMPLIED. ALL DIMENSIONS ARE SUBJECT TO NORMAL MANUFACTURING VARIATIONS.

Amphenol
Sine Systems - www.amphenol-sine.com
44724 Morley Drive
Clinton Township, MI 48036

A series, NMEA 2000 Multi-Hub Adapter

SIZE: B | TYPE: C- | DWG NO: AM-N04PX2-HA | REVISION: A2
SCALE: NONE | SHEET 1 OF 1

TITLE: A series, NMEA 2000 Multi-Hub Adapter | DWG NO: AM-N04PX2-HA | REV: A2 | SH: 1 | OF: 1

https://www.amphenol-sine.com/pdf/datasheet/AM-N04PX2-HA.pdf

Yamaha Engine Interface Cable for NMEA 2000

and many other brand specific adapters.

https://www.amazon.com/JustMarineCables-Yacht-Yamaha-Engine-Interface/dp/B093TV824M

Backbone Cables and Drop Tees

- Yellow connectors are for the backbone
  - Up to 250 meters
- Black connector are “drop” cables for devices

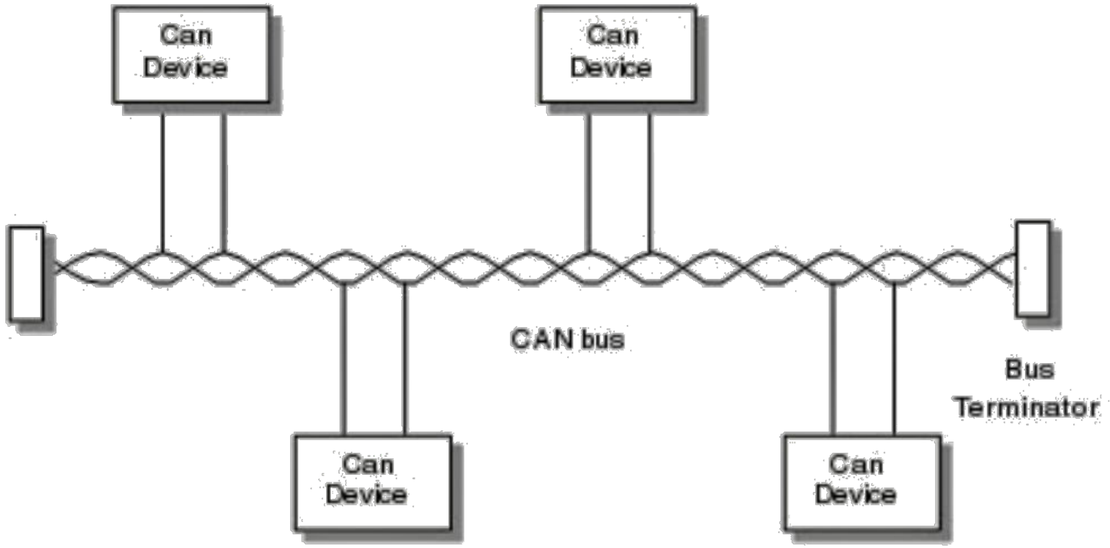
We’ll use this backbone cable for connecting to friends later.

Power Tee

- Connect red to + and black to –
- Low power devices only (4 amps)
  - Stereos and engines have their own power

Terminating Resistors

- 120 ohms each
- Installed at the end of the backbone
- Necessary for clean signaling
  - Reduces signal reflections
  - Provides current path for strong signals
- 60 ohms overall



https://1.bp.blogspot.com/-vDpMSdE5ViE/VM3OfB-IGRI/AAAAAAAAROM/iIrei4KxDQg/s1600/Controller-Area-Network.jpg

PEAK CAN Adapter

- Requires PCAN Drivers installed in Windows
- Linux Kernel drivers already built-in

Example Sensors

- Temperature
- Fluid Level
  - Fresh Water
  - Fuel
  - Black Water
- Weather
- GPS
- Sonar
- Speed

Complete Individual Setup

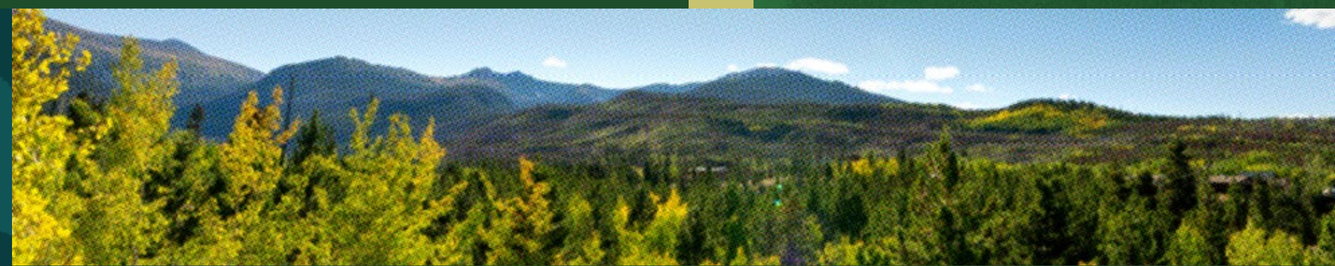
Exercise: Connecting to Your Own CAN

Proposed process:

1. Connect Red Power Tap to 4-way Backbone Tee
2. Connect Terminating Resistors
3. Connect PEAK to Dsub-to-M12 cable
4. Connect M12 cable to Backbone
5. Connect Sensor to Backbone
6. Connect power
7. Plug in PEAK USB adapter.

Controller Area Network Signaling

What is on the CAN Bus wires?



Controller Area Network

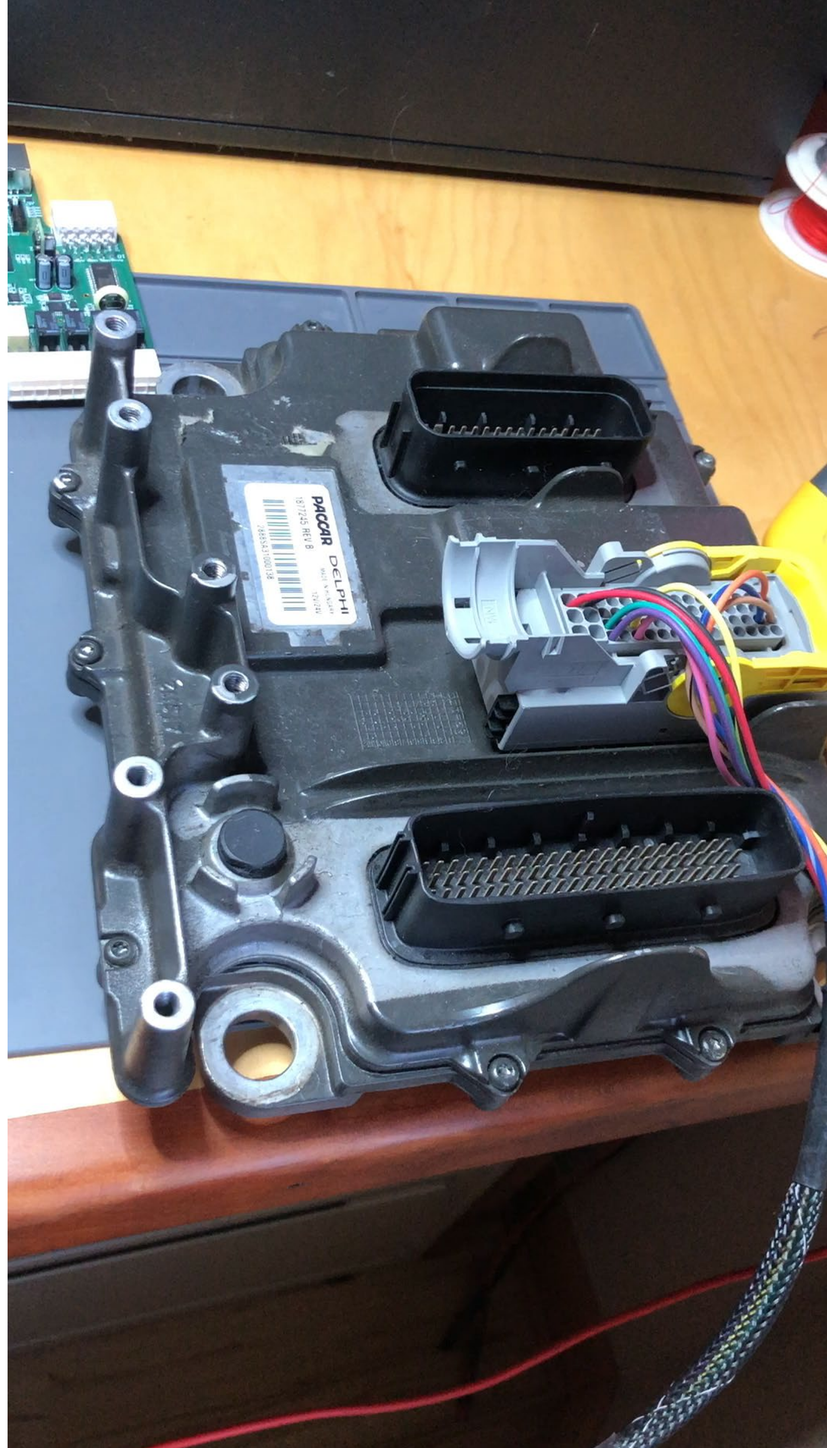
- Introduced by Bosch in the 1980s
- Multi-master priority-based bus access with non-destructive message arbitration
- Utilizes a 15-bit cyclic redundancy check (CRC) to reliably detect transmission errors
- Reliable delivery is built in with an acknowledgement bit at the end of the frame
- Low latency with up to 8 bytes of data per frame (Classic CAN)
- Bit rates up to 1mbit/second
- Required on all passenger cars for emission compliance starting in 2008 (Standard 11-bit CAN ID)
- Utilized by SAE J1939 as the foundational networking protocol in the 1990s

Extended 29-bit CAN Frame

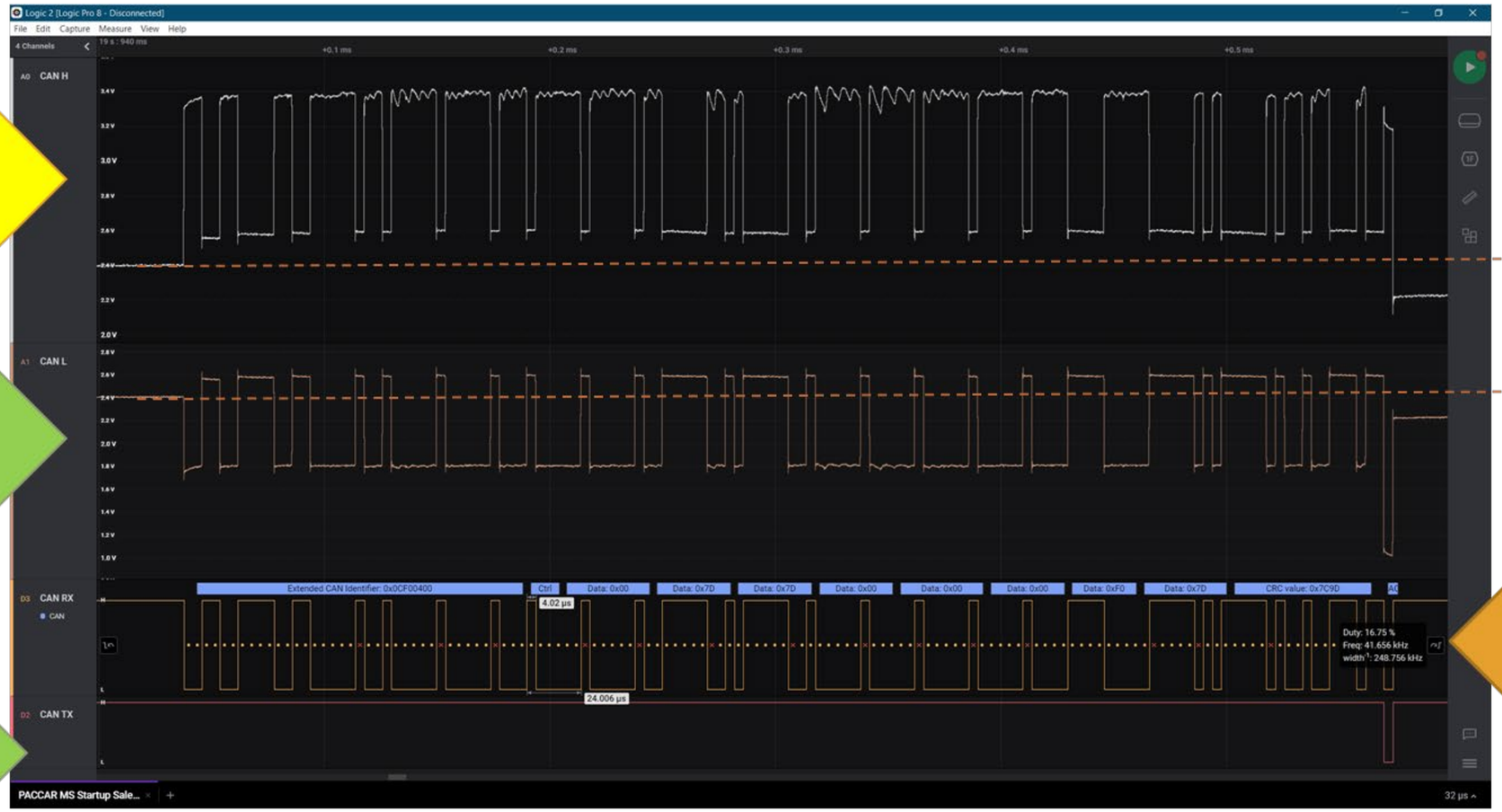
SOF	11-bit CAN ID	SRR	IDE	18-bit CAN ID	RTR	Control Field	Data Bytes	CRC	ACK	EOF

CAN Signaling: Measurement Example

- PACCAR MX Engine Control Module (ECM)
- Synercon Technologies Smart Sensor Simulator
  - Completes the CAN network circuit
  - Provides connectivity for the ECM
- DG Technologies J1939 Breakout Box
- Raspberry Pi with a CAN-FD Hat
  - Runs embedded Linux with SocketCAN
  - Records CAN traffic using can-utils `candump` command
- Fluke Scope Meter as an Oscilloscope
  - Measures voltage traces between CAN High and CAN Low
- Saleae Logic Probe
  - Analog Voltage measurements (duplicating the oscilloscope)
  - Digital measurements from the CAN Transceiver
  - CAN signal decoding features
  - PC application interface



CAN Signaling: Single Frame



CAN High

CAN Low

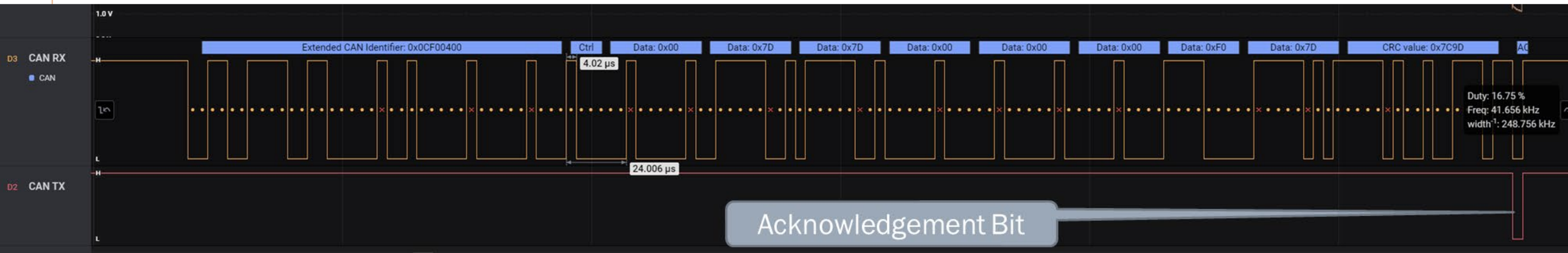
Transceiver TX

Each line is at 2.4V for a quiet bus.

Digital logic from Transceiver RX

CAN Measurement Observations

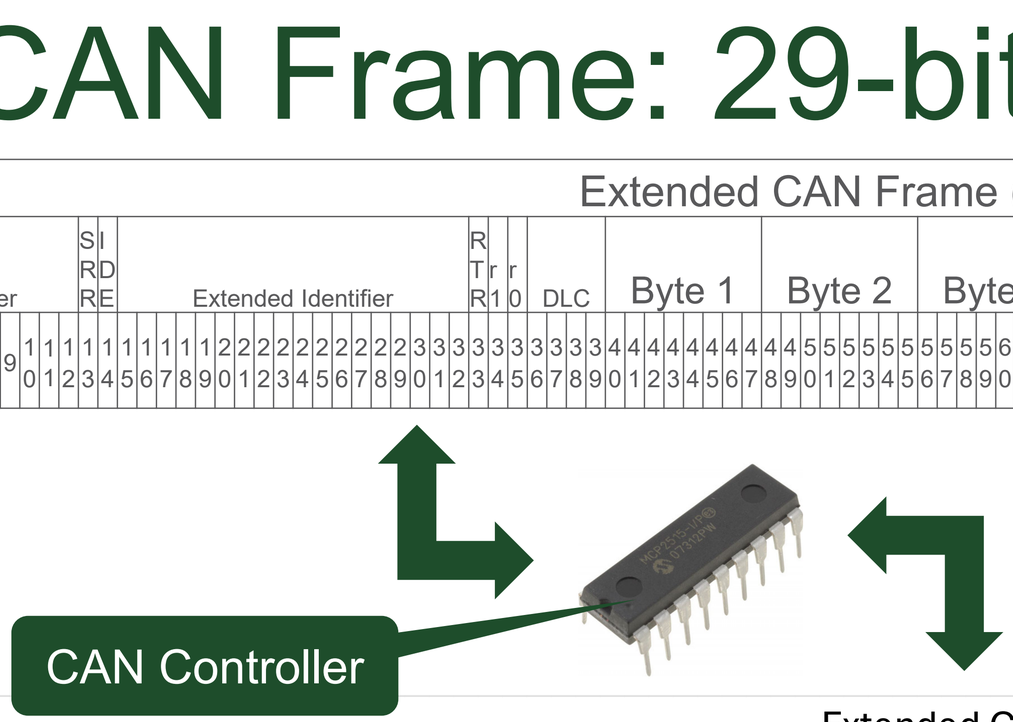
- A bit time is about 4 microseconds. This is 1/250000 seconds.
- Data that has all zeros still has extra bits in the field. These are called stuff bits.
- Stuff bits are inserted after 5 sequential bits of the same value.
- At the end of the message, A bit is seen on the TX line, which indicates an acknowledgement the message was received.
- The total message length is about 500us.
- Signaling is non-return-to-zero (NRZ).



CAN Frame: 29-bit ID and up to 8 bytes

Extended CAN Frame (no stuff bits) in the Transceiver

	SOF	Identifier	SRR	IDE	Extended Identifier	RTR	r1	r0	DLC	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7	Byte 8	Cyclic Redundancy Check	CRC	ACK	End of Frame
CAN Bit Num	1	2 3 4 5 6 7 8 9 10 11 12	13	14	15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32	33	34	35	36 37 38 39	40 41 42 43 44 45 46 47	48 49 50 51 52 53 54 55	56 57 58 59 60 61 62 63	64 65 66 67 68 69 70 71	72 73 74 75 76 77 78 79	80 81 82 83 84 85 86 87	88 89 90 91 92 93 94 95	96 97 98 99 100 101 102 103	104 105 106 107 108 109 110 111 112 113 114 115 116 117 118 119	120	121	122 123 124 125 126 127 128



Extended CAN Frame in Application

	Identifier	DLC	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7	Byte 8
CAN Bit Num	2 3 4 5 6 7 8 9 10 11 12 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32	36 37 38 39	40 41 42 43 44 45 46 47	48 49 50 51 52 53 54 55	56 57 58 59 60 61 62 63	64 65 66 67 68 69 70 71	72 73 74 75 76 77 78 79	80 81 82 83 84 85 86 87	88 89 90 91 92 93 94 95	96 97 98 99 100 101 102 103

Exercise: Collect CAN Data

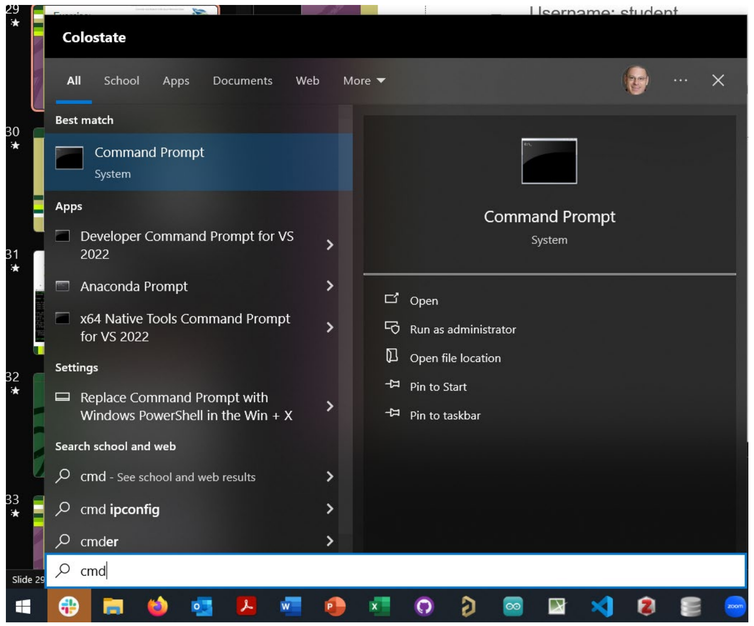
- Login to the computers
  - Username: student
  - Password: student
- Open (and edit) the 01_ReadCAN.py file
  - Use socketcan for Linux
  - Use pcan for Windows
- python-can uses the same code for both Linux and Window once connected.

01_ReadCAN.py > ...

```python
#!/usr/bin/python
# To install python-can in Linux either
#   sudo apt install python3-can
# or
#    pip install python-can
import can

# Configure the CAN interface for Linux:
# Determine the right can channel from the terminal:
#     dmesg | grep can
# From the terminal, do the following:
#     lsmod | grep can
# This should show can_dev as peak_usb. If not,
# load the kernel module with this command:
#     sudo modprobe can can_raw can_dev
# Run this for each CAN Channel:
#     sudo ip link set can0 up type can bitrate 250000
# Check to see it was installed:
#     ifconfig
# This should show the installed CAN channel.
device = "socketcan"
channel = 'can2'

# To configure the CAN interface for Windows,
# uncommment these two lines
device = 'pcan'           # The PCAN Drivers must be installed in Windows
channel = 'PCAN_USBBUS1'  # Update this to your specific channel

bitrate = 250000          # Set the bitrate to 2500000 for all NMEA2000

#connect to the controller area network
bus = can.interface.Bus(channel=channel, interface=device, bitrate=bitrate)

#read some messages
for i in range(20):
    msg = bus.recv(timeout=1.0)
    data_string = " ".join(["{:02X}".format(b) for b in msg.data])
    print(f"{i:3d} {msg.arbitration_id:08X} [{msg.dlc}] {data_string}")
bus.shutdown()
print(f"Finished reading {i} CAN messages.")
```

Exercise: Collect CAN Data

- Login to the computers
  - Username: student
  - Password: student
- Run the following command line program:
  - `python 01_ReadCAN.py`



Controller Area Network (CAN) Quick Reference Guide

CyberBoat Challenge℠

SocketCAN in Linux

https://www.kernel.org/doc/html/latest/networking/can.html

#	Task or Goal	Terminal Command	Outcomes and notes
1	Load the CAN kernel modules	`sudo modprobe can can_dev can_raw`	The Kernel modules are loaded, but there is no terminal output.

2	Check to see the kernel modules are loaded	`lsmod | grep can`	`jdaily@metalbox:~/Documents $ lsmod | grep can`
			`can_raw 20480 0`
			`can 24576 1 can_raw`
			`can_dev 45056 2 mcp251x,peak_usb`

3	Determine CAN channels available	`dmesg | grep can`	`jdaily@metalbox:~/Documents $ dmesg | grep can`
			`[ 5.395028] mcp251x spi1.2 can0: MCP2515 successfully initialized.`
			`[ 5.435872] mcp251x spi1.1 can1: MCP2515 successfully initialized.`
			`[45631.010207] peak_usb 1-1.1:1.0 can2: attached to PCAN-USB FD channel`

4	Bring CAN channel up and set bitrate	`sudo ip link set can0 up type can bitrate 250000`	No terminal output
5	Bring CAN channel down	`sudo ip link set can0 down`	No terminal output Do this before setting new bitrate

6	Check Status of CAN channel with details and statistics	`ip -d -s link show can0`	`jdaily@metalbox:~/Documents $ ip -d -s link show can2`
			`6: can2: <NOARP,UP,LOWER_UP,ECHO> mtu 16 qdisc pfifo_fast state UP mode DE`
			`link/can promiscuity 0 allmulti 0 minmtu 0 maxmtu 0`
			`can state ERROR-ACTIVE (berr-counter tx 0 rx 0) restart-ms 0`
			`bitrate 250000 sample-point 0.875`
			`tq 25 prop-seg 69 phase-seg1 70 phase-seg2 20 sjw 10 brp 2`
			`pcan_usb_fd: tseg1 1..256 tseg2 1..128 sjw 1..128 brp 1..1024 br`
			`pcan_usb_fd: dtseg1 1..32 dtseg2 1..16 dsjw 1..16 dbrp 1..1024 d`
			`clock 80000000`
			`re-started bus-errors arbit-lost error-warn error-pass bus-off`
			`0 0 0 0 0 0`
			`ax_segs 65535 gro_max_size 65536 parentbus usb parentdev 1-1.1:1.0`
			`RX: bytes packets errors dropped missed mcast`
			`100327 12685 0 0 0 0`
			`TX: bytes packets errors dropped carrier collsns`
			`0 0 0 0 0 0`
			`jdaily@metalbox:~/Documents $`

7	Install can-utils	`Sudo apt install can-utils`	`jdaily@metalbox:~/Documents $ sudo apt install can-utils`
			`Reading package lists... Done`
			`Building dependency tree... Done`
			`Reading state information... Done`
			`can-utils is already the newest version (2020.11.0-1).`

8	Show CAN Traffic	`candump any`	`jdaily@metalbox:~/Documents $ candump any`
			`can1 15FD0618 [8] 3E 41 76 FF FF FF FF FF`
			`can1 15FD0718 [8] 3E C0 41 76 FF 7F FF FF`
			`can2 15FD0618 [8] 3E 41 76 FF FF FF FF FF`
			`can2 15FD0718 [8] 3E C0 41 76 FF 7F FF FF`

9	Log 100 frames using hardware timestamps	`candump -H -l -e -n 100 can0`	`jdaily@metalbox:~/Documents $ candump -H -l -e -n 100 can0`
			`Disabled standard output while logging.`
			`Enabling Logfile 'candump-2024-12-08_091232.log'`

10	Fuzz	`cangen -g0 -i -e -L8 can0`	Random extended CAN frames transmitted
11	Denial of Service	`cangen -g0 -i -e -D00 -L8 -I0 -v can0`	All zeros are transmitted

12	Display CAN	`cansniffer -c -t0 -h200 can0`	`57|ms | -- ID -- | data ... < can0 # l=20 h=200 t=0 slots=4 >`
			`01003 | 0DF50B18 | 5E FF FF FF FF 00 00 FF ^.....`
			`00500 | 15FD0618 | 5D 41 76 FF FF FF FF FF ]Av.....`
			`00501 | 15FD0718 | 5D C0 41 76 FF 7F FF FF ].Av....`
			`02004 | 15FD0818 | 51 00 00 41 76 00 00 Q..Av..`

13	Show Busload	`canbusload -t -b -r -e can0@250000`	`canbusload 2024-12-08 09:41:45 (exact bitstuffing)` `can0@250000 483 72397 30904 28% |XXXXX.....|`
14	Send a single can message	`cansend can0 18EEFF18#00.00.00.00.00.00.00.00`	A single message is transmitted

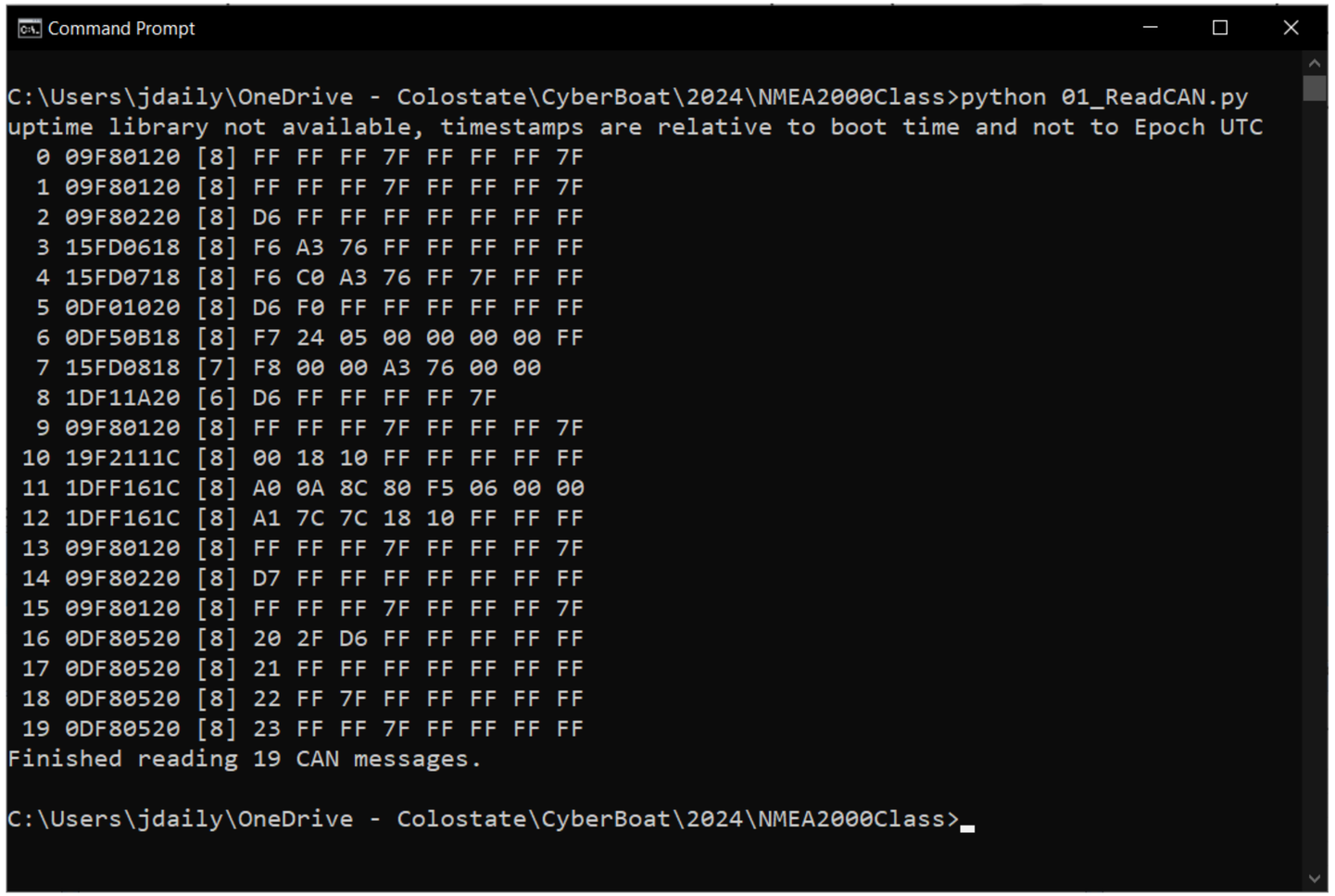
15	Run program with python-can	`python3 01_ReadCAN.py`	`jdaily@metalbox:~/Documents $ python 01_ReadCAN.py`
			`0 15FD0618 [8] 0C 41 76 FF FF FF FF FF`
			`1 15FD0718 [8] 0C C0 41 76 FF 7F FF FF`
			`2 0DF50B18 [8] 0D FF FF FF FF 00 00 FF`
			`3 15FD0818 [7] 0E 00 00 41 76 00 00`

Tip: If you get a “write: No buffer space available” message, then bring the channel down and up again.

Tip: keep a cansniffer terminal always open to monitor the bus.

```
jdaily@metalbox:~ $ cangen -g0 can1
write: No buffer space available
jdaily@metalbox:~ $ sudo ip link set can1 down
jdaily@metalbox:~ $ sudo ip link set can1 up
jdaily@metalbox:~ $ cangen -g0 -i can1
^C
Counted 6430914 ENOBUFS return values on write().
```


SYSTEMS ENGINEERING
COLORADO STATE UNIVERSITY



CAN Data in the Terminal

This can be duplicated using candump in Linux.

How can we send CAN messages?

Sending CAN w/ can-utils in Linux

- `cansend can0 18EAFFF9#00EE00`

```
jdaily@mastercraft: ~
jdaily@mastercraft:~ $ cansend
cansend - send CAN-frames via CAN_RAW sockets.

Usage: cansend <device> <can_frame>.

<can_frame>:
 <can_id>#{data}          for 'classic' CAN 2.0 data frames
 <can_id>#R{len}          for 'classic' CAN 2.0 data frames
 <can_id>##<flags>{data}  for CAN FD frames

<can_id>:
 3 (SFF) or 8 (EFF) hex chars
{data}:
 0..8 (0..64 CAN FD) ASCII hex-values (optionally separated by '.')
{len}:
 an optional 0..8 value as RTR frames can contain a valid dlc field
<flags>:
 a single ASCII Hex value (0 .. F) which defines canfd_frame.flags

Examples:
  5A1#11.2233.44556677.88 / 123#DEADBEEF / 5AA# / 123##1 / 213##311223344 /
  1F334455#1122334455667788 / 123#R / 00000123#R3

jdaily@mastercraft:~ $
```

```
jdaily@mastercraft: ~
jdaily@mastercraft:~ $ cangen
cangen - CAN frames generator.

Usage: cangen [options] <CAN interface>
Options:
         -g <ms>       (gap in milli seconds - default: 200 ms)
         -e            (generate extended frame mode (EFF) CAN frames)
         -f            (generate CAN FD CAN frames)
         -b            (generate CAN FD CAN frames with bitrate switch (BRS))
         -E            (generate CAN FD CAN frames with error state (ESI))
         -R            (send RTR frame)
         -m            (mix -e -f -b -E -R frames)
         -I <mode>     (CAN ID generation mode - see below)
         -L <mode>     (CAN data length code (dlc) generation mode - see below)
         -D <mode>     (CAN data (payload) generation mode - see below)
         -p <timeout>  (poll on -ENOBUFS to write frames with <timeout> ms)
         -n <count>    (terminate after <count> CAN frames - default infinite)
         -i            (ignore -ENOBUFS return values on write() syscalls)
         -x            (disable local loopback of generated CAN frames)
         -c            (number of messages to send in burst, default 1)
         -v            (increment verbose level for printing sent CAN frames)

Generation modes:
 'r'         => random values (default)
 'i'         => increment values
 <hexvalue>  => fix value using <hexvalue>

When incrementing the CAN data the data length code minimum is set to 1.
CAN IDs and data content are given and expected in hexadecimal values.

Examples:
cangen vcan0 -g 4 -I 42A -L 1 -D i -v -v
        (fixed CAN ID and length, inc. data)
cangen vcan0 -e -L i -v -v -v
        (generate EFF frames, incr. length)
cangen vcan0 -D 11223344DEADBEEF -L 8
        (fixed CAN data payload and length)
cangen vcan0 -g 0 -i -x
        (full load test ignoring -ENOBUFS)
```

`cangen -g 250 -e -I 18EAFFF9 -D EBFE00 -L 3 can0`

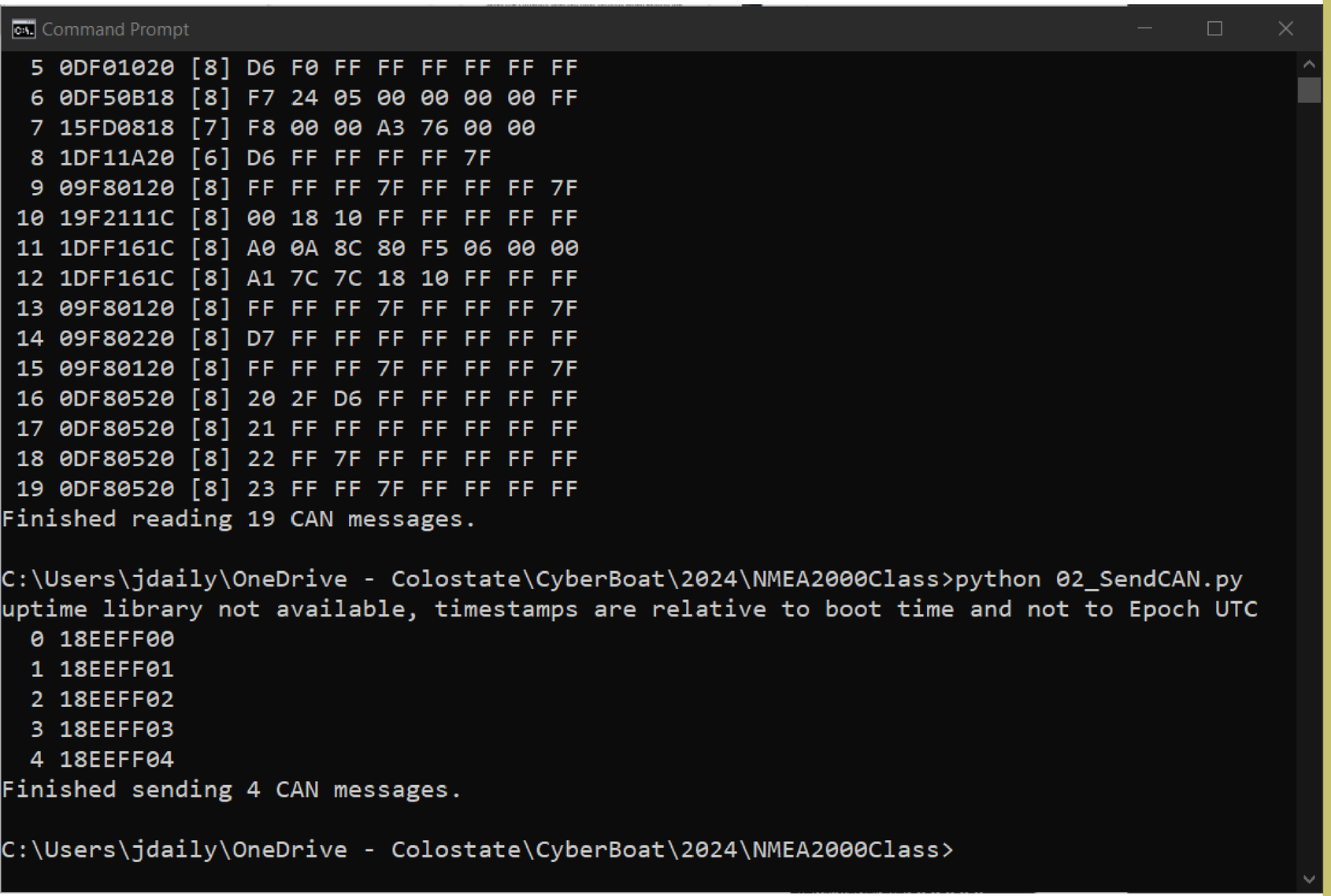
Exercise: Send CAN Data

Run the following command line program:

```
python 02_SendCAN.py
```

02_SendCAN.py > ...

```python
#!/usr/bin/python
import can
import time
import sys

if 'win' in sys.platform:
    device = 'pcan'           # The PCAN Drivers must be installed in Windows
    channel = 'PCAN_USBBUS1'  # Update this to your specific channel
else:
    device = 'socketcan'
    channel = 'can0'
bitrate = 250000   # Set the bitrate to 2500000 for all NMEA2000
#connect to the controller area network
bus = can.interface.Bus(channel=channel, interface=device, bitrate=bitrate)

#make an address claim message
address_claim_message = can.Message(
    arbitration_id=0x18EEFF00,
    is_extended_id=True,
    data=[0x00]*8,
    check=True
)

#Send address claim messages to claim all the addresses.
for i in range(5):
    msg = bus.send(address_claim_message)
    print(f"{i:3d} {address_claim_message.arbitration_id:08X}")
    address_claim_message.arbitration_id += 1 #increment the source address by 1
    time.sleep(.260)
```



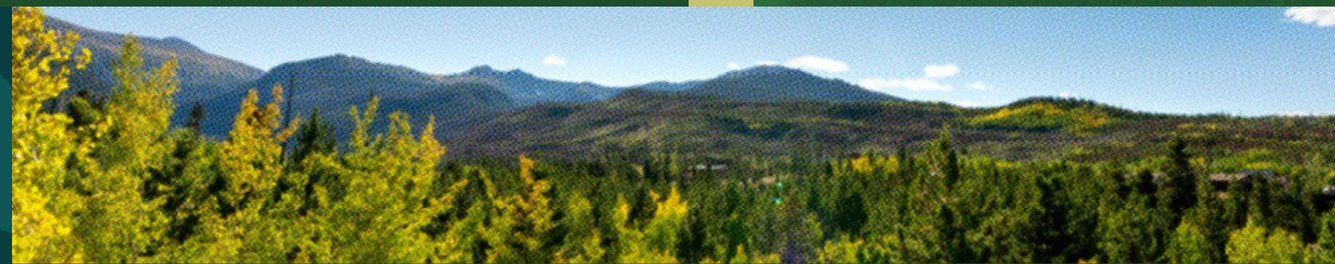
Send Data Example

How do we know if this worked?

(Future Exercise)

Creating Meaning from Messages

How do we get engineering values from CAN messages according to J1939 and NMEA2000?



SAE J1939 is Built on CAN

The main features that define J1939 are:

- A standardized meaning for 29-bit arbitration identifiers.
- A mechanism for sending messages larger than 8 bytes (up to 1785 bytes) using the transport protocol.
- The ability for a controller application to negotiate a unique source address.

SAE International	SURFACE VEHICLE RECOMMENDED PRACTICE	SAE J1939 JUN2012
		Issued 2000-04 Revised 2012-06 Superseding J1939 APR2011
Serial Control and Communications Heavy Duty Vehicle Network - Top Level Document		

J1939 Network Layers

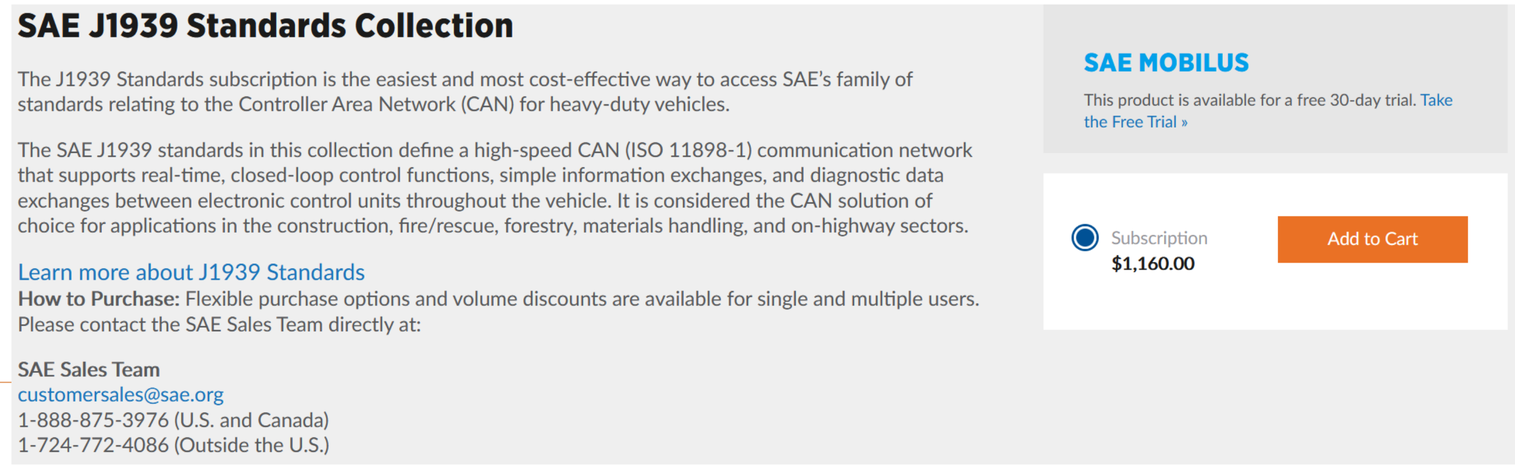
Layer	Name	Standard(s)	Description
9	Security	J1939-91	Cybersecurity considerations for J1939
8	Management	J1939-81	Address Claiming and network management
7	Application	J1939-71 J1939-73	Applications (-71) and diagnostics (-73) explaining the conversion of bits and bytes to engineering units
6	Presentation	Not Used These services are built into the Data Link Layer.	
5	Session		
4	Transport		
3	Network	J1939-31	Network management and gateways
2	Data Link	J1939-2X	Transport Protocol, PDUs, Structure
1	Physical	J1939-1X	Wiring, Signaling, Connectors

SAE J1939 Standards Organization

- Follows the OSI 7-layer model for naming, e.g.:
  - J1939-7X are for application layers
  - J1939-1X are for physical layers
- The standard collection adds much more definition to the CAN communications
- Includes additional “Layers”
  - J1939-8X Network Management
  - J1939-9X Network Security
- J1939 is large and not free

Recommendation:
- Acquire the Digital Annex first.
- Read J1939-21 for details on the PDU

- J1939 Accommodates Extensions
  - PGN 0xEF00 is Proprietary A
  - PGN 0xFFXX is Proprietary B
  - PGN 0xDA00 is ISO-15765 (UDS)
- A Digital Annex (J1939DA) has the applications defined in an Excel spreadsheet



SAE J1939 Standards Collection

The J1939 Standards subscription is the easiest and most cost-effective way to access SAE’s family of standards relating to the Controller Area Network (CAN) for heavy-duty vehicles.

The SAE J1939 standards in this collection define a high-speed CAN (ISO 11898-1) communication network that supports real-time, closed-loop control functions, simple information exchanges, and diagnostic data exchanges between electronic control units throughout the vehicle. It is considered the CAN solution of choice for applications in the construction, fire/rescue, forestry, materials handling, and on-highway sectors.

Learn more about J1939 Standards

How to Purchase: Flexible purchase options and volume discounts are available for single and multiple users. Please contact the SAE Sales Team directly at:

SAE Sales Team
customersales@sae.org
1-888-875-3976 (U.S. and Canada)
1-724-772-4086 (Outside the U.S.)

SAE MOBILUS

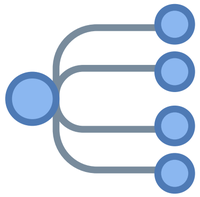
This product is available for a free 30-day trial. Take the Free Trial »

Subscription
$1,160.00

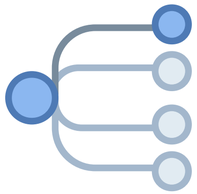
Add to Cart

Types of J1939 Messages

- Broadcast
  - Messages sent for any controller application (CA) to use
  - CAN Arbitration ID specifies a source address (SA) in the last 8 bits of the ID
  - Implicitly defined the destination address as 255 (Global)



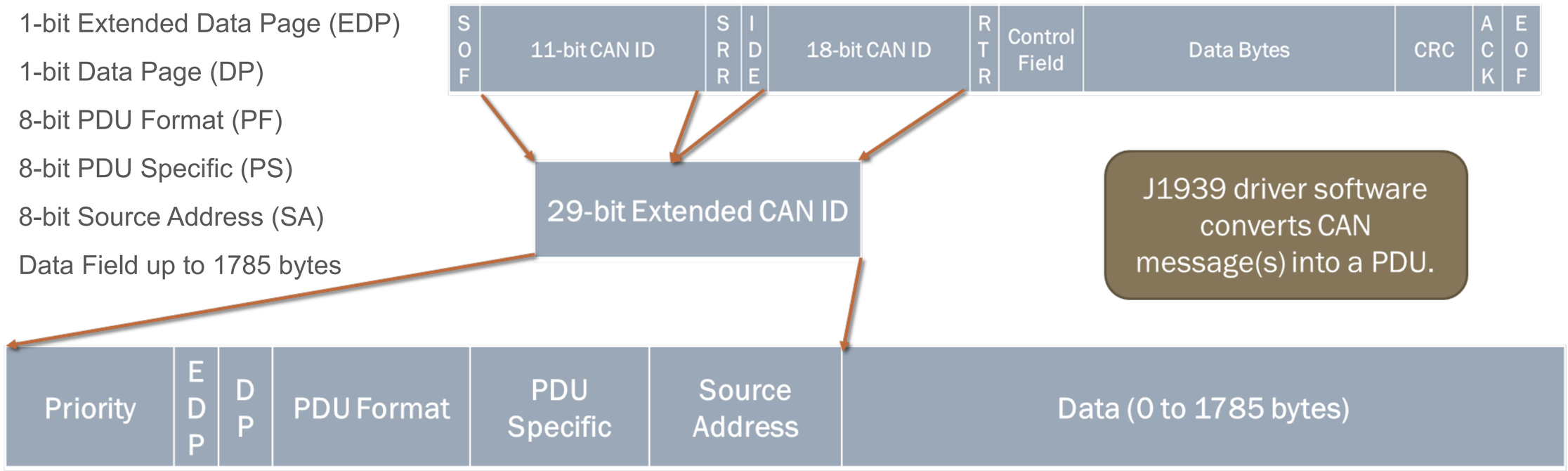
- Point-to-Point
  - One controller application sends a message to another
  - CAN Arbitration ID specifies a source address (SA) in the last 8 bits of the ID
  - A destination address (DA) is in bits 9-16 of the CAN ID.



How do some messages implicitly set the destination address to 255?

J1939 Protocol Data Unit

- A J1939 message has all the elements in the protocol data unit (PDU)
  - 3-bit Priority
  - 1-bit Extended Data Page (EDP)
  - 1-bit Data Page (DP)
  - 8-bit PDU Format (PF)
  - 8-bit PDU Specific (PS)
  - 8-bit Source Address (SA)
  - Data Field up to 1785 bytes

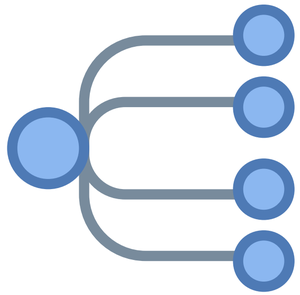


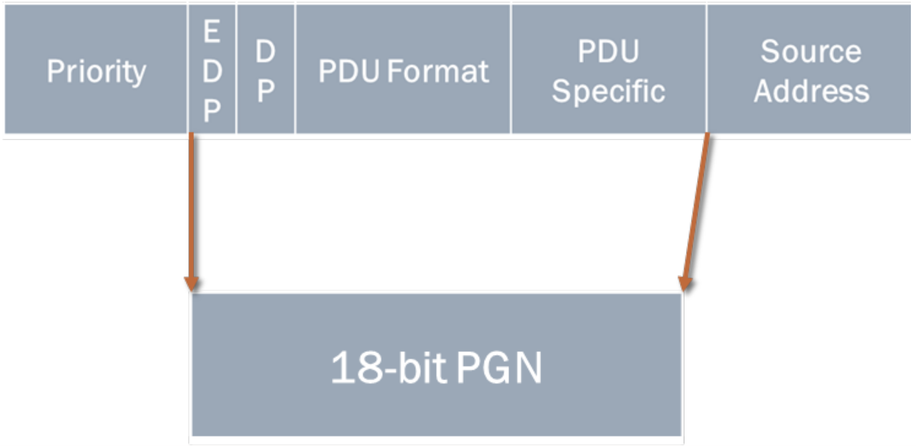
PDU 2 Format Messages

- The PDU format #2 is for broadcast messages
  - EDP, DP, PF and PS create the 18-bit Parameter Group Number (PGN)
  - PS becomes the group extension (GE)
  - PF value must be greater than or equal to 240 (0xF0)
  - Destination address is implied to be 255 (0xFF)
- Parameter Groups Numbers are 18 bits.
  - Most applications on the boat (NMEA 2000) set the EDP to zero and DP to 1
  - Parameter Groups collect similar data for the PDU data field
  - PDU 2 messages have a hex values where the leading nibble is F

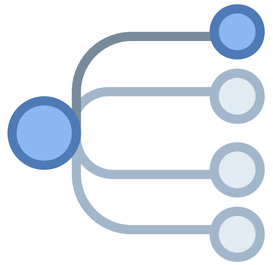
Examples:

  - PGN 61444 (0x0F004) for the Electronic Engine Controller 1 group
  - PGN 129025 (0x1F801) is for Position, Rapid Update

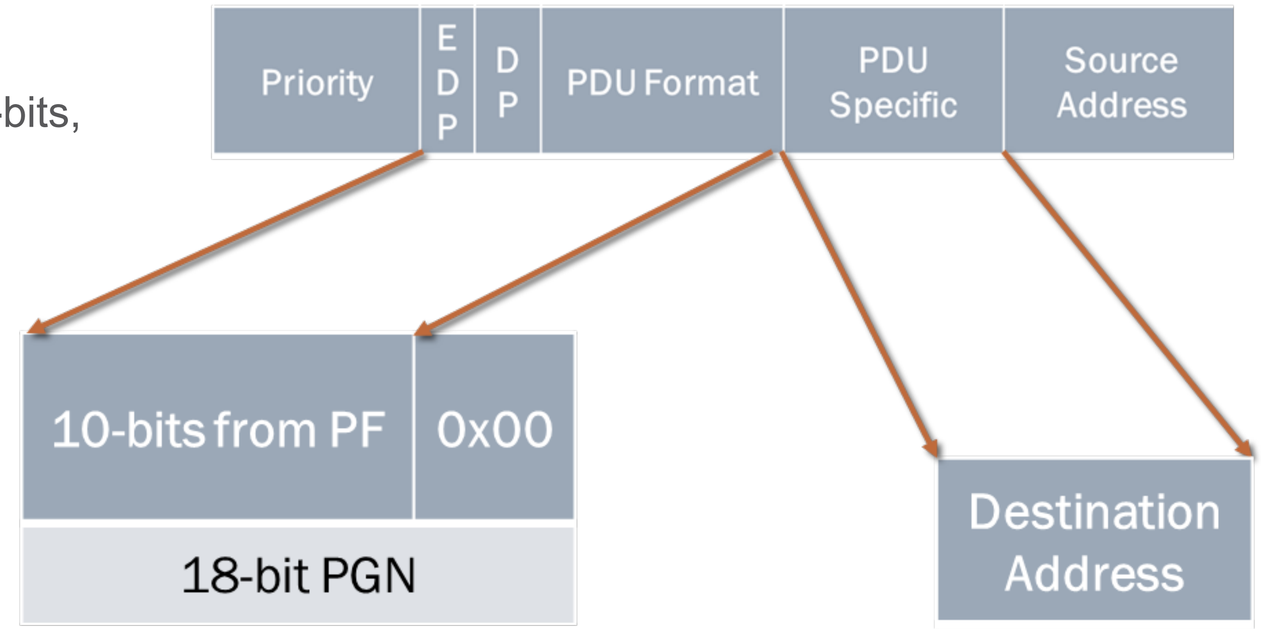




PDU 1 Format Messages



- The PDU format #1 is for point-to-point messages
  - EDP, DP, PF and 00 create the Parameter Group Number (PGN)
  - PS becomes the destination address (DA)
- Parameter Group Numbers (PGNs) are still 18-bits, but the last 8 bits are set to zero.
- Source and Destination are explicit
- PGN values in hex do not have 0xF as the first nibble.

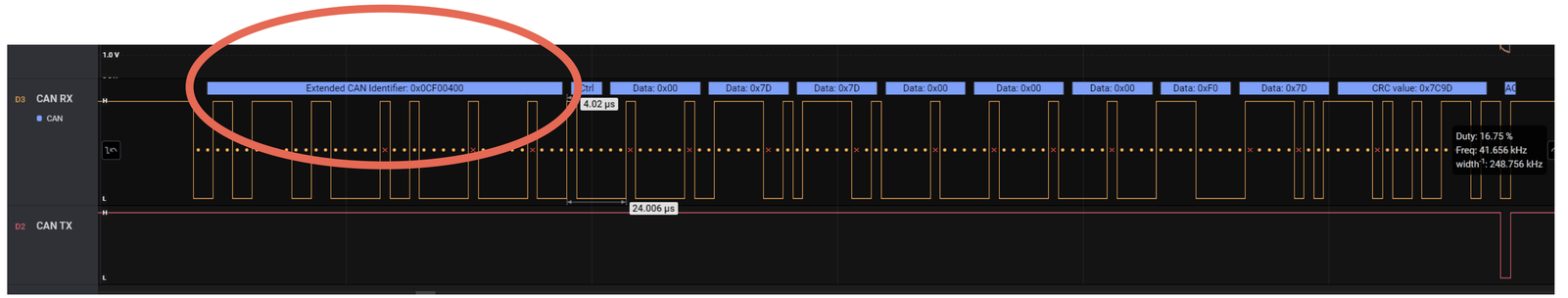


Processing CAN IDs

1. Read the CAN ID as a 32-bit integer
2. Separate the ID into the PDU elements using bit masking and bit shifting
3. Determine if it is a PDU1 or PDU2 message based on the value of PF
   a) If PDU1, PS is the Destination Address
   b) If PDU2, PS is the Group Extension, set Destination Address to 0xFF

```
PRIORITY_MASK = 0x1C000000
EDP_MASK      = 0x02000000
DP_MASK       = 0x01000000
PF_MASK       = 0x00FF0000
PS_MASK       = 0x0000FF00
SA_MASK       = 0x000000FF
PDU1_PGN_MASK = 0x03FF0000
PDU2_PGN_MASK = 0x03FFFF00
```

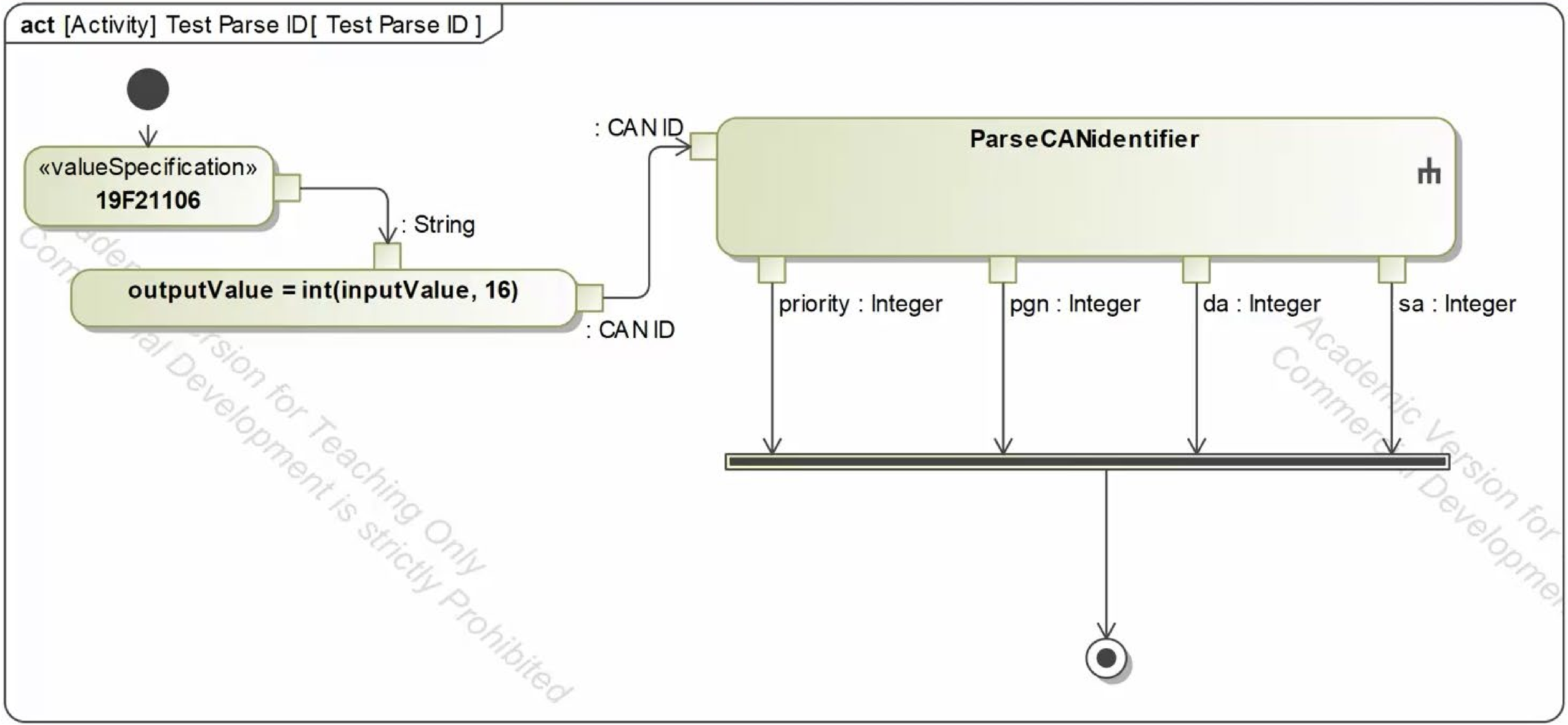
Bit Masking and Shifting



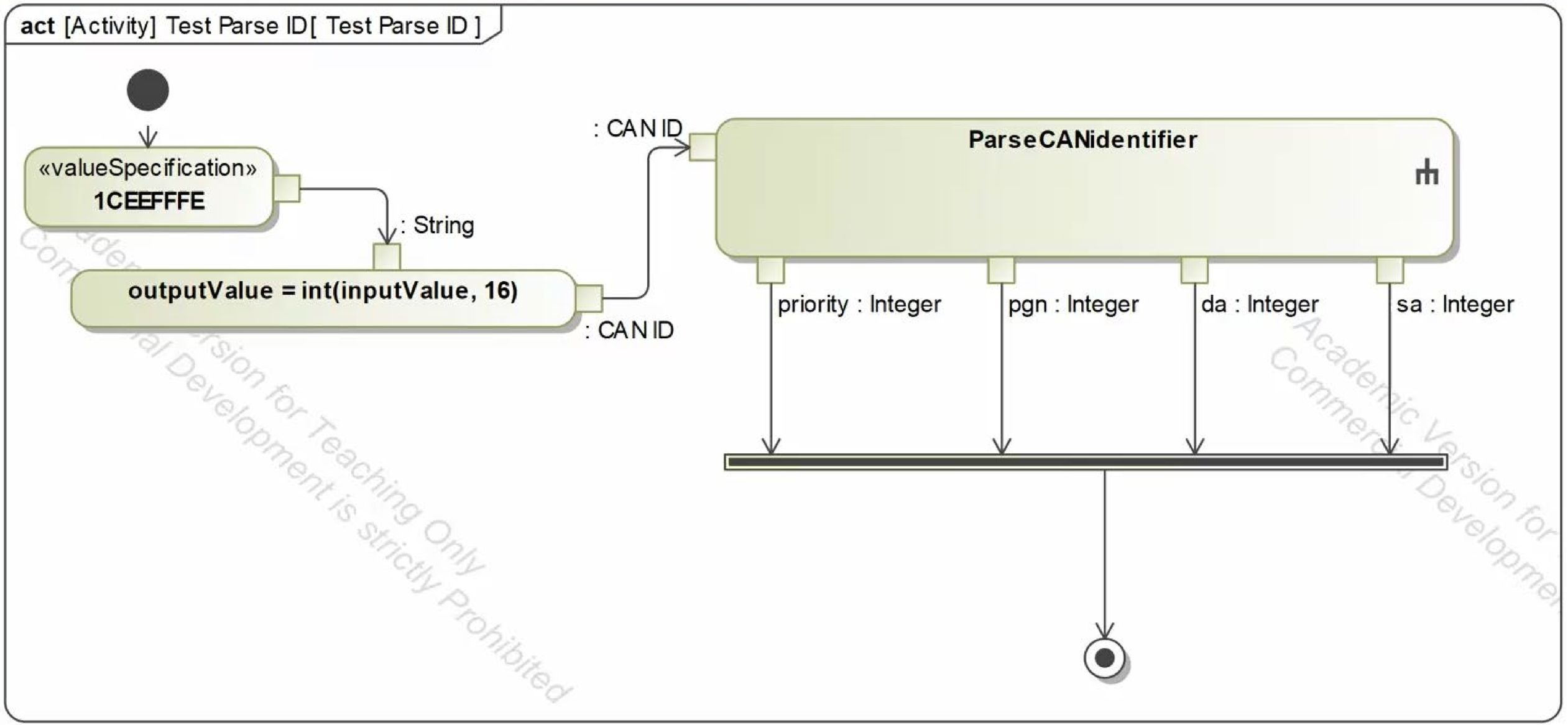
Example: Determine the PGN from the CAN Frame Capture

ID Hex Nibbles	0	C				F				0				0				4				0				0			
ID Binary	0	1	1	0	0	1	1	1	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
PGN Mask Hex	0	3				F				F				F				F				0				0			
Mask Binary	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0
AND Result	0	0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
Shift right 8 bits									0	0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	0	0	1	0	0
Hex after shift									0	0				F				0				0				4			

0x00F004 = 61444 dec



Parsing PDU Format 2 (Broadcast)



Parsing PDU Format 1 (Point-to-Point)

03_ReadJ1939.py > ...

```python
#!/usr/bin/python
import can
import struct
from cyberboat import *

#connect to the controller area network
device, channel, bitrate = selectCANsystem()
bus = can.interface.Bus(channel=channel,
                        interface=device, bitrate=bitrate)

def parseCANid(id):
    priority =  (0x1C000000 & id) >> 26
    sa = 0xFF & id #Source Address
    PF = (0xFF0000 & id) >> 16 #PDU Format
    if PF < 0xF0: #Type 1 message
        da = (0xFF00 & id) >> 8 #Destination Address
        pgn = (0x3FF0000 & id) >> 8 #Parameter Group Number (18 bits)
    else: #Type 2 Message
        da = 0xFF #Global destination (any node)
        pgn = (0x3FFFF00 & id) >> 8 #Parameter Group Number (18 bits)
    return (priority, pgn, da, sa)

#Receive some messages
for i in range(25):
    msg = bus.recv(timeout=1.0)
    if msg is None:
        continue
```

Exercise: Parse CAN into J1939 Protocol Data Unit (PDU)

Data from the CAN ID include:

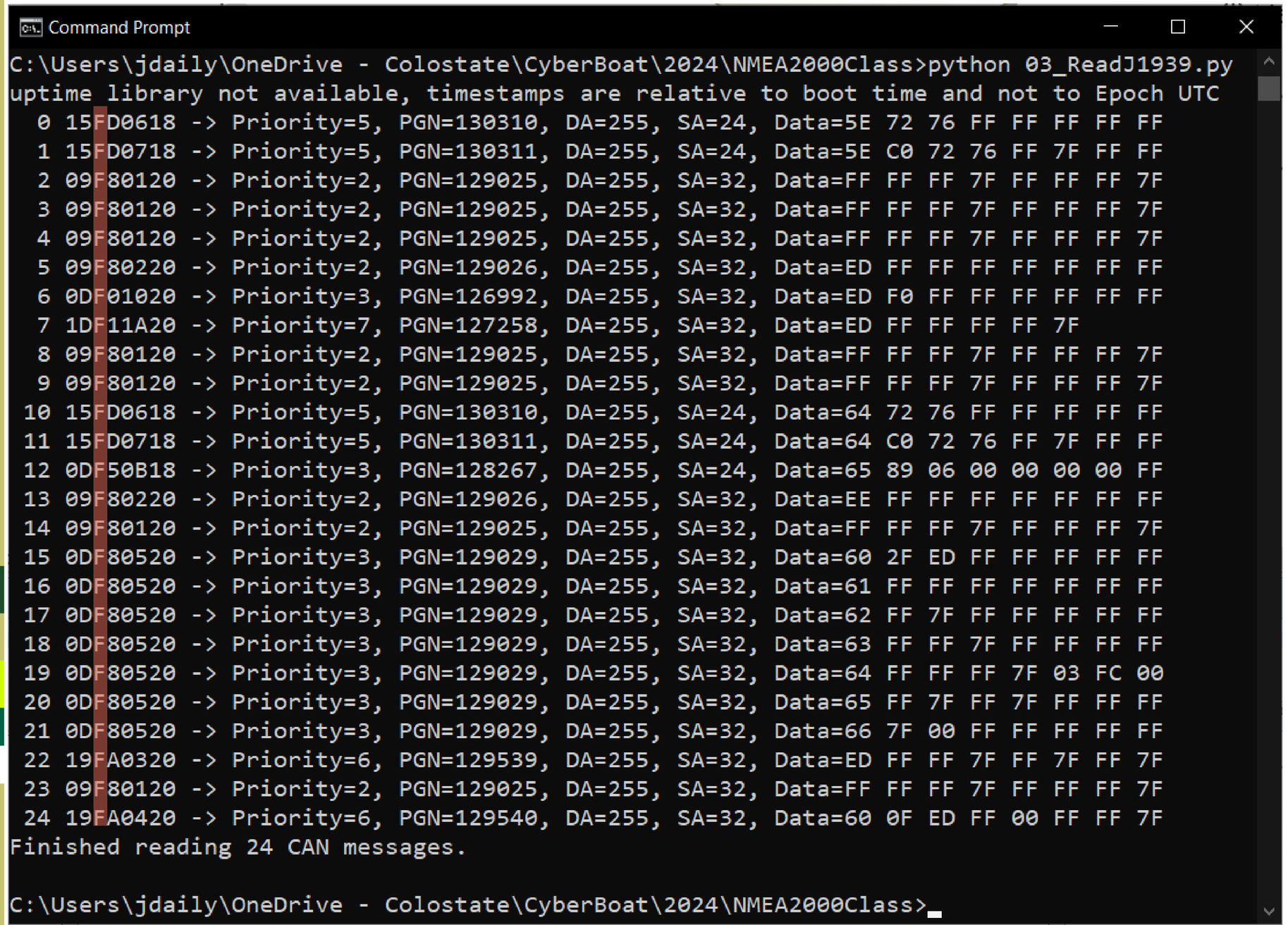
- Priority
- Parameter group number
- Destination Address
- Source Address

Run `python 03_ReadJ1939.py`

How many PDU2 Format messages are there?

What's the ratio of PDU1 vs PDU2?

Parsing the ID Results



All these example messages are PDU2 (Broadcast). The first PGN nibble is 0x0F = 240.

NATIONAL MARINE ELECTRONICS ASSOCIATION

NMEA 2000 ®

STANDARD FOR SERIAL-DATA NETWORKING OF MARINE ELECTRONIC DEVICES

Appendix A

Application Layer
(Parameter Group Definitions)

Version 3.000
February 2022

Determine Message Contents from PGN

15FD0618 -> Priority=5, PGN=130310, DA=255, SA=24, Data=21 11 76 FF FF FF FF FF

Look up the PGN in Appendix A of the NMEA 2000 Standard

Environmental Parameters - DEPRECATED **PGN: 130310 hex: 1FD06**

This PGN has been deprecated (as of version 1.200, PGN 130311 replaced PGN 130310) and is not recommended for new designs. However, support of PGN 130310 may be necessary to ensure compatibility with legacy equipment. PGN 130311 has also been deprecated as of version 1.210. The following PGNs are recommended to be used for new designs: 130314-Actual Pressure, 130315-Set Pressure, 130316-Temperature-Extended Range. The latest definition of PGN 130310 before deprecation was as follows: Local atmospheric environmental conditions.

Single Frame: Yes *Priority Default:* 5 *Default Update Rate:* 500 milliseconds *Frequency:* 2. cycles per second
Destination: Global *Query Support:* Optional *Command Support:* Optional *ACK Rqmnts:* None

Field #	Field Name					
1	**Sequence ID**		*Byte Field Size:* 1		*Request Parameter*	Optional
			Bit Field Size:		*Command Parameter:*	Optional
	DD056 Sequence ID		An upward counting number that binds information transmitted in two or more PGNs from a single source address. Identical SID values within two or more different PGN transmissions identifies those PGN transmissions as a single related data set. For example, identical SID values bind the COG and SOG values in PGN 129026 to the Latitude and Longitude values in PGN 129029 as a single data set. 0 - 252 = binding available (when SID value reaches 252, resume with 0 on next data set) 253 - 254 = reserved for future use 255 = No binding provided. NMEA recommends using binding SID values whenever practical.			
	DF53 Integer, 8 bit unsigned	**uint8**	*Range:* 0 to 252	*Resolution:* 1 bit	Unit-less number	
2	**Water Temp**		*Byte Field Size:* 2		*Request Parameter*	Optional
			Bit Field Size:		*Command Parameter:*	Optional
	DD043 Generic Temperature					
	DF39 Temperature, low	**uint16**	*Range:* 0 to 655.32 deg K	*Resolution:* 1x10E-2 deg K		
3	**Outside Ambient Air Temp.**		*Byte Field Size:* 2		*Request Parameter*	Optional
			Bit Field Size:		*Command Parameter:*	Optional
	DD043 Generic Temperature					
	DF39 Temperature, low	**uint16**	*Range:* 0 to 655.32 deg K	*Resolution:* 1x10E-2 deg K		
4	**Atmospheric Pressure**		*Byte Field Size:* 2		*Request Parameter*	Optional
			Bit Field Size:		*Command Parameter:*	Optional
	DD049 Generic Pressure					
	DF47 Pressure, medium	**uint16**	*Range:* 0 to 6,553,200 Pa	*Resolution:* 1x10E+2 Pa		
5	**NMEA Reserved**		*Byte Field Size:*		*Request Parameter*	
			Bit Field Size: resv 8		*Command Parameter:*	
	DD001 Reserved field		Variable number of reserved bits, all set to logic "1"			
	DF52 Bit field	**bit(n)**	*Range:* Variable	*Resolution:* 1	Used to construct bit fields	
	Used to align subsequent data on a byte boundary.					

Environmental Parameters

Decode the data: 21 11 76 FF FF FF FF FF

Field 1 (1 byte): Sequence ID
0x51 = 81

Field 2 (2 Bytes): Water Temperature in deg K
0x7611 = 302.25 deg K = 84.34 deg F

Fields 3 – 5 are Unavailable (0xFF)

Temp Sensor

Sensor producing data

SAE J1939 Digital Annex

Used to Interpret

- Source Addresses
- Parameter Group Numbers
- Address NAME fields
- Engineering Units (SLOT)

What's in a PG?

NMEA 2000 Fields

AIS Interrogation **PGN: 129803 hex: 1FB0B**

This parameter group provides data associated with the ITU-R M.1371 Message 15 Interrogation Message used to request a specific ITU-R M.1371 message resulting in responses from one or more AIS mobile units. An AIS device may generate this parameter group either upon receiving a VHF data link message 15, or upon receipt of an ISO or NMEA request PGN. The Command Group Function PGN 126208 shall be used with this PGN to configure base station interrogation parameters (see ITU-R M.1371-1 for additional information). Note that future revisions to the ITU-R M.1371 VHF Data Link Messages may result in their spare or reserved bits being defined with a specific meaning, requiring the spare or reserved parameter in this parameter group to have the corresponding new meaning in future revisions of this standard.

Single Frame: No *Priority Default:* 7 *Default Update Rate:* milliseconds *Frequency:* NA cycles per second
Destination: Global *Query Support:* Optional *Command Support:* Required *ACK Rqmnts:* None

Field #	Field Name				
1	**Message ID**		*Byte Field Size:* *Bit Field Size:* 6	*Request Parameter* *Command Parameter:*	Optional Required
	DD188 AIS Message Identifier		Message Identifier (range of 0 to 63). See the latest version of ITU-R M.1371 for more information.		
	DF52 Bit field	**bit(n)**	*Range:* Variable	*Resolution:* 1	Used to construct bit fields
	15 = Interrogation Message				
2	**Repeat Indicator**		*Byte Field Size:* *Bit Field Size:* 2	*Request Parameter* *Command Parameter:*	Optional Note 1
	DD185 AIS Repeater Indicator		Used by the repeater to indicate how many times a message has been repeated (range of 0 to 3). 0 = Default 1 = First retransmission 2 = Second retransmission 3 = Final retransmission See the latest version of ITU-R M.1371 for more information.		
	DF52 Bit field	**bit(n)**	*Range:* Variable	*Resolution:* 1	Used to construct bit fields
3	**Source ID**		*Byte Field Size:* 4 *Bit Field Size:*	*Request Parameter* *Command Parameter:*	Optional Required
	DD010 Generic numeric ID, large		Number of route, waypoint, event, mark, etc.		
	DF55 Integer, 32 bit unsigned	**uint32**	*Range:* 0 to 4,294,967,292	*Resolution:* 1 bit	Unit-less number
	MMSI number of interrogating station.				
4	**NMEA Reserved**		*Byte Field Size:* *Bit Field Size:* resv 1	*Request Parameter* *Command Parameter:*	
	DD001 Reserved field		Variable number of reserved bits, all set to logic "1"		
	DF52 Bit field	**bit(n)**	*Range:* Variable	*Resolution:* 1	Used to construct bit fields
	Used to align subsequent data on byte boundary.				

J1939 Suspect Parameters

PGN 61444 ***Electronic Engine Controller 1*** ***EEC1***

Engine related parameters

Transmission Repetition Rate: engine speed dependent
Data Length: 8
Extended Data Page: 0
Data Page: 0
PDU Format: 240
PDU Specific: 4 PGN Supporting Information:
Default Priority: 3
Parameter Group Number: 61444 (0x00F004)

Start Position	Length	Parameter Name	SPN
1.1	4 bits	Engine Torque Mode	899
1.5	4 bits	Actual Engine - Percent Torque High Resolution	4154
2	1 byte	Driver's Demand Engine - Percent Torque	512
3	1 byte	Actual Engine - Percent Torque	513
4-5	2 bytes	Engine Speed	190
6	1 byte	Source Address of Controlling Device for Engine Control	1483
7.1	4 bits	Engine Starter Mode	1675
8	1 byte	Engine Demand – Percent Torque	2432

SPN 190 ***Engine Speed***

Actual engine speed which is calculated over a minimum crankshaft angle of 720 degrees divided by the number of cylinders.

Data Length: 2 bytes
Resolution: 0.125 rpm/bit, 0 offset
Data Range: 0 to 8,031.875 rpm Operational Range: same as data range
Type: Measured
Supporting Information:
PGN reference: 61444

Data Decoding and Encoding: Meaning for Bits and Bytes

- Common data sizes
  - Bit Mapped, like Switch States, (2-bits)
  - Single Byte Data (8-bits)
  - 2-byte Data (16 bits)
  - 4-byte Data (32 bits)
  - ASCII data (variable)

Scale, Limits, Offsets, Transfer (SLOTs)

Identifier	SLOT Name	SLOT Type	Scaling	Range	Offset	Length
1	SAEpr11	Pressure	5 kPa/bit	0 to 1,250 kPa	0	1 byte
2	SAEpr13	Pressure	8 kPa/bit	0 to 2,000 kPa	0	1 byte
3	SAEtm11	Time	1 h/bit	0 to 250 h	0	1 byte
4	SAEtm10	Time	1 h/bit	-125 to 125 h	-125 h	1 byte
5	SAEtm12	Time	1 h/bit	-32,127 to 32,128 h	-32,127 h	2 bytes
6	SAEtm06	Time	1 s/bit	0 to 4,211,081,215 s	0	4 bytes
7	SAEad01	Angle/Direction	0.0000001 deg/bit	-210 to 211.1081215 deg	-210 deg	4 bytes
⋮	⋮	⋮	⋮	⋮	⋮	⋮

Decoding Example: Accelerator Pedal Low Idle Switch

- Given the following CAN message (hex):
  0CF00300 [8] D0 5A 25 FF FF 0F A0 81
- Break the CAN ID into J1939 values
  - 0x0C is priority 3
  - F003 is PDU2 format
    - PGN is 0xF003 = 61443, Electronic Engine Control 2
    - Destination Address is 0xFF (implied)
  - Source address is 0x00 = 0, Engine #1

- Determine some Suspect Parameters in the data
  - 558: Accelerator Pedal 1 Low Idle Switch
  - 559: Accelerator Pedal 1 Kickdown Switch
  - 1437: Road Speed Limit Status
  - 2970: Accelerator Pedal 2 Low Idle Switch
  - 91: Accelerator Pedal Position 1
  - 92: Engine Percent Load At Current Speed
  - 974: Remote Accelerator Pedal Position
  - 29: Accelerator Pedal Position 2
  - 2979: Vehicle Acceleration Rate Limit Status
  - 3357: Actual Maximum Available Engine % Torque
  - 5398: Estimated Pumping – Percent Torque

Bit 8

Bit 1

Byte Position	Byte 1				Byte 2						Byte 8
Hex Values	0xD0				0x5A					0	0x81
Binary	0b 1101 0000										
SPNs	2970	1437	559	558	91	92	974	29	2979		
Engineering	N/A	On	Off	Off			N/A	N/A			

Bit Encoding for SPN 558
00 - Accelerator pedal 1 not in low idle condition
01 - Accelerator pedal 1 in low idle condition
10 - Error
11 - Not available

Decoding Example: Accelerator Pedal Position

- Given the following CAN message (hex):
  0CF00300 [8] D0 5A 25 FF FF 0F A0 81
- Break the CAN ID into J1939 values
  - 0x0C is priority 3
  - F003 is PDU2 format
    - PGN is 0xF003 = 61443, Electronic Engine Control 2
    - Destination Address is 0xFF (implied)
  - Source address is 0x00 = 0, Engine #1

- Determine some Suspect Parameters in the data
  - 558: Accelerator Pedal 1 Low Idle Switch
  - 559: Accelerator Pedal 1 Kickdown Switch
  - 1437: Road Speed Limit Status
  - 2970: Accelerator Pedal 2 Low Idle Switch
  - 91: Accelerator Pedal Position 1
  - 92: Engine Percent Load At Current Speed
  - 974: Remote Accelerator Pedal Position
  - 29: Accelerator Pedal Position 2
  - 2979: Vehicle Acceleration Rate Limit Status
  - 3357: Actual Maximum Available Engine % Torque
  - 5398: Estimated Pumping – Percent Torque

Byte Position	Byte 1				Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7	Byte 8
Hex Values	0xD0				0x5A	0x25	0xFF	0xFF	0x0F	0xA0	0x81
Binary	0b 1101 0000										
SPNs	2970	1437	559	558	91	[illegible]2	974	29	2979		
Engineering	N/A	On	Off	Off	90*0.4 = 36%		N/A	N/A			

Byte order for Integers (Endianness)

Post office boxes have mail loaded from the inside and taken out through the front.

Mailboxes have mail loaded and removed from the front.

Byte order for Integers (Endianness)

- Let's pretend our mailboxes are containers to hold bytes representing integers.
- Example: the integer 100,293,486 is 0x05FA5B6E in hex, which is 4 bytes:

Most Significant Byte

`0x05 0xFA 0x5B 0x5E`

Least Significant Byte

- Pretend each byte is a small package. In what order should the postmaster insert the bytes into the mailbox so they are in order when the customer extracts them?

MSB first

`0x05 0xFA 0x5B 0x5E`

Big Endian

LSB first

`0x05 0xFA 0x5B 0x5E`

Little Endian

Byte order for Integers (Endianness)

Endianness doesn't affect single byte integers.

- SAE J1939 encodes multi-byte integers in the Little Endian format.
- This give the appearance the bytes are reversed and need to be swapped to interpret
- Intel format = Little Endian = Least Significant Byte first
- Motorola format = Big Endian = Most Significant Byte first (as we typically read and write) are used for CAN IDs

Byte Length	Decimal	Hex	Big Endian	Little Endian (J1939)
1	241	F1	0xF1	0xF1
2	743	2E7	0x02 0xE7	0xE7 0x02
2	25	19	0x00 0x19	0x19 0x00
4	1,890,056,399	70A7F8CF	0x70 0xA7 0xF8 0xCF	0xCF 0xF8 0xA7 0x70

Decoding Example: Engine Speed (RPM)

- Given the following CAN message (hex):
  `0CF00400 [8] 31 9D 9D A2 38 00 0F 9D`
- Break the CAN ID into J1939 values
  - 0x0C is priority 3
  - F004 is PDU2 format
    - PGN is 0xF004 = 61444, Electronic Engine Control 1
    - Destination Address is 0xFF (implied)
  - Source address is 0x00 = 0, Engine #1

- Determine some Suspect Parameters in the data
  - 889: Engine Torque Mode
  - 4154: Actual Percent Torque
  - 512: Driver's Demand Engine - Percent Torque
  - 513: Actual Engine Percent Torque
  - 190: Engine Speed
  - 1483: SA of Controlling device
  - 1675: Engine Starter Mode
  - 2432: Engine Demand- Percent Torque

Byte Position	Byte 1		Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7	Byte 8
Hex Values	0x31		0x9D	0x9D	0xA2	0x38	0x00	0x0F	0x9D
SPNs	899	4154	512	513	190		1483	1675	2432
Engineering			32%	32%	14,498/8=1812.25				32%

AutoSave Off | J1939DA JAN23.xlsx • Last Modified: Yesterday at 7:48 PM | Search (Alt+Q) | Daily,Jeremy

File Home Insert Page Layout Formulas Data Review View Help Acrobat | Comments | Share

Cut Copy Format Painter | Arial 10 | Wrap Text | General | Merge & Center | Conditional Formatting | Format as Table | Normal 2 | Normal 3 | Normal_Com... | Note 2 | Normal | Bad | Good | Neutral | Calculation | Check Cell | Insert | Delete | Format | AutoSum | Fill | Clear | Sort & Filter | Find & Select | Analyze Data | Sensitivity

Clipboard | Font | Alignment | Number | Styles | Cells | Editing | Analysis | Sensitivity

D13 =C13/8

	A	B	C	D	E
1	0CF00400 [8] 31 9D 9D A2 38 00 0F 9D				
2					
3	Priority	0C	3		
4	PGN	F004	61444		
5	DA	FF	255		
6	SA	00	0		
7					
8	Engine Speed, SPN 190				
9	Byte4	A2			
10	Byte5	38			
11					
12	SPN190	Hex	Dec	RPM	
13		38A2	14498	1812.25	rev/min

Abbreviations | Revision Column Definition | SAE J1939DA Schema Changes | SPs & PGs | Sheet2 | SLOTs | Manufacturer IDs (B10) | Global Source Addresses (B2) | Global NAME Func ...

Ready | Accessibility: Investigate | 175%

8:07 PM 6/24/2023

04_DecodeJ1939.py > ...

```
for msg in [msg1, msg2, msg3]:
    # msg = bus.recv(timeout=1.0) #use this line if the bus is connected
    data_string = " ".join([f"{b:02X}" for b in msg.data])
    (priority, pgn, da, sa) = parseCANid(msg.arbitration_id)
    print(f"{msg.arbitration_id:08X} -> Priority={priority}, PGN={pgn}, DA={da},

    key = f"{pgn}_{sa}"
    decoded_data[key] = {}
    if pgn == 61444: #J1939 Digital Annex
        decoded_data[key]["PG Label"]="Electronic Engine Controller 1"
        decoded_data[key]["PG Acronym"]="EEC1"
        decoded_data[key]["PGN"]=pgn
        decoded_data[key]["SPs"]={}

        decoded_data[key]["SPs"][889]={
            "SP Label":"Engine Torque Mode",
            "value": msg.data[0] & 0x0F,
            "SLOT": 89,
            "range": "0 to 15"
        }

        decoded_data[key]["SPs"][4154]={
            "SP Label":"Actual Engine - Percent Torque (Fractional)",
            "value": (msg.data[0] & 0xF0) >> 4,
            "SLOT": 268,
            "range":"0 to 1.625"
        }

        decoded_data[key]["SPs"][512]={
            "SP Label":"Driver's Demand Engine - Percent Torque",
            "value": msg.data[1] - 125,
            "SLOT": 45,
            "range":"-125% to 125%"
        }
```

Exercise: Determine Meaning

Decode the Following Messages:

`0CF00400 FF 7D 7D 70 12 00 FF FF`

`09F8021C 0E FC 24 8D A8 00 FF FF`

`09F20000 00 54 09 FF FF FF FF FF`

Edit and run 04_DecodeJ1939.py

Use the NMEA2000 PDF and the J1939DA Excel

J1939 Digital Annex Entry for EEC1

SPs and PGs

All SP and PG assignments. The complete technical definition details are provided for most of the SPs and PGs. If the technical definition details for an SP or a PG are not provided on this tab, then refer to the

Some information in cells may not be visible due to the size of the cell. Expansion of the cell, particularly in the description columns, may be necessary to view the complete contents.

Re

	Re	PGN	PG Label	PG Acronym	PG Description	PDU Type	PGN Page	PG Data Length Class	PG Data Minimum Length	PG Data Length Constraint	PG Data Highest Assigned Byte	Transmission Rate	Default Priority
1851		61444	Electronic Engine Controller 1	EEC1	Engine related parameters	PDU2	Page 0	Fixed	8		8	Fixed rate of 10 to 50 ms or	3
1852		61444	Electronic Engine Controller 1	EEC1	Engine related parameters	PDU2	Page 0	Fixed	8		8	Fixed rate of 10 to 50 ms or	3
1853		61444	Electronic Engine Controller 1	EEC1	Engine related parameters	PDU2	Page 0	Fixed	8		8	Fixed rate of 10 to 50 ms or	3
1854		61444	Electronic Engine Controller 1	EEC1	Engine related parameters	PDU2	Page 0	Fixed	8		8	Fixed rate of 10 to 50 ms or	3
1855		61444	Electronic Engine Controller 1	EEC1	Engine related parameters	PDU2	Page 0	Fixed	8		8	Fixed rate of 10 to 50 ms or	3
1856		61444	Electronic Engine Controller 1	EEC1	Engine related parameters	PDU2	Page 0	Fixed	8		8	Fixed rate of 10 to 50 ms or	3
1857		61444	Electronic Engine Controller 1	EEC1	Engine related parameters	PDU2	Page 0	Fixed	8		8	Fixed rate of 10 to 50 ms or	3
1858		61444	Electronic Engine Controller 1	EEC1	Engine related parameters	PDU2	Page 0	Fixed	8		8	Fixed rate of 10 to 50 ms or	3

	SP Position in PG	SP Start Bit	SPN	SP Label	SP Description	SP Length	Scaling	Offset	Data Range	Operational Range	Unit	SLOT Identifier
1851	1.1	1.1	899	Engine Torque Mode	State signal which indicates which engine torque mode is currently	4 bits	16 states	0	0 to 15			89
1852	1.5	1.5	4154	Actual Engine - Percent Torque (Fractional)	This parameter displays an additional torque in percent of the	4 bits	0.125 % per bit	0 %	0 to 1.625 %	0.0 to 0.875 %	%	268
1853	2	2.1	512	Driver's Demand Engine - Percent Torque	The requested torque output of the engine by the driver. It is based on	1 byte	1 % per bit	-125 %	-125 to 125 %	0 to 125%	%	45
1854	3	3.1	513	Actual Engine - Percent Torque	The calculated output torque of the engine. The data is transmitted in	1 byte	1 % per bit	-125 %	-125 to 125 %	0 to 125%	%	45
1855	4-5	4.1	190	Engine Speed	Actual engine speed which is calculated over a minimum crankshaft	2 bytes	0.125 rpm per bit	0 rpm	0 to 8 031.875 rpm		rpm	76
1856	6	6.1	1483	Source Address of Controlling Device for Engine Control	The source address of the SAE J1939 device currently controlling the	1 byte	1 source address pe	0 source addr	0 to 255 source address	0 to 253	source addre	35
1857	7.1	7.1	1675	Engine Starter Mode	There are several phases in a starting action and different reasons	4 bits	16 states	0	0 to 15			89
1858	8	8.1	2432	Engine Demand – Percent Torque	The requested torque output of the engine by all dynamic internal	1 byte	1 % per bit	-125 %	-125 to 125 %	-125 to 125%	%	45

EEC1 Message Decoding

```
0CF00400 FF 7D 7D 70 12 00 FF FF
```


```
"61444_0": {
    "PG Acronym": "EEC1",
    "PG Label": "Electronic Engine Controller 1",
    "PGN": 61444,
    "SPs": {
        "190": {
            "SLOT": 76,
            "SP Label": "Engine Speed",
            "range": "0 to 8031.875 rpm",
            "value": 590.0
        },…
        "512": {
            "SLOT": 45,
            "SP Label": "Driver's Demand Engine - Percent Torque",
            "range": "-125% to 125%",
            "value": 0
        },
        "513": {
            "SLOT": 45,
            "SP Label": "Actual Engine - Percent Torque",
            "range": "-125% to 125%",
            "value": 0
        },
```

COG & SOG, Rapid Update

PGN: 129026
hex: 1F802

This PGN is a single frame PGN that provides Course Over Ground (COG) and Speed Over Ground (SOG). Being a single frame message, as opposed to other PGNs that include COG and SOG and are defined as multi-packet, this PGN lends itself to being transmitted more frequently, without using up excessive bandwidth on the bus. This may be of benefit to receiving equipment requiring rapid COG and SOG updates.

Single Frame: Yes *Priority Default:* 2 *Default Update Rate:* 250 milliseconds *Frequency:* 4. cycles per second
Destination: Global *Query Support:* Optional *Command Support:* Optional *ACK Rqmnts:* None

Field #	Field Name					
1	**Sequence ID**		*Byte Field Size:* 1 *Bit Field Size:*		*Request Parameter* Optional *Command Parameter:* Optional	
	DD056 Sequence ID		An upward counting number that binds information transmitted in two or more PGNs from a single source address. Identical SID values within two or more different PGN transmissions identifies those PGN transmissions as a single related data set. For example, identical SID values bind the COG and SOG values in PGN 129026 to the Latitude and Longitude values in PGN 129029 as a single data set. 0 - 252 = binding available (when SID value reaches 252, resume with 0 on next data set) 253 - 254 = reserved for future use 255 = No binding provided. NMEA recommends using binding SID values whenever practical.			
	DF53 Integer, 8 bit unsigned	**uint8**	*Range:* 0 to 252	*Resolution:* 1 bit	Unit-less number	
2	**COG Reference**		*Byte Field Size:* *Bit Field Size:* 2		*Request Parameter* Optional *Command Parameter:* Optional	
	DD117 Direction reference		0 = True, 1 = Magnetic, 2 = Error, 3 = Null			
	DF52 Bit field	**bit(n)**	*Range:* Variable	*Resolution:* 1	Used to construct bit fields	
3	**NMEA Reserved**		*Byte Field Size:* *Bit Field Size:* resv 6		*Request Parameter* *Command Parameter:*	
	DD001 Reserved field		Variable number of reserved bits, all set to logic "1"			
	DF52 Bit field	**bit(n)**	*Range:* Variable	*Resolution:* 1	Used to construct bit fields	
	Used to align subsequent data on a byte boundary.					
4	**Course Over Ground**		*Byte Field Size:* 2 *Bit Field Size:*		*Request Parameter* Optional *Command Parameter:* Optional	
	DD165 Course-Over-Ground (COG)		The direction of the path over ground actually followed by a vessel.			
	DF02 Angle	**uint16**	*Range:* 0 to 2Pi rad	*Resolution:* 1x10E-4 rad	Resolution ~0.0057deg, 1 deg = .01745 rad	
5	**Speed Over Ground**		*Byte Field Size:* 2 *Bit Field Size:*		*Request Parameter* Optional *Command Parameter:* Optional	
	DD044 Generic Speed					
	DF35 Speed	**uint16**	*Range:* 0 to 655.32 m/s	*Resolution:* 1x10E-2 m/s	1 Knot = 0.5144 m/s	

NMEA 2000 Entry for 129026

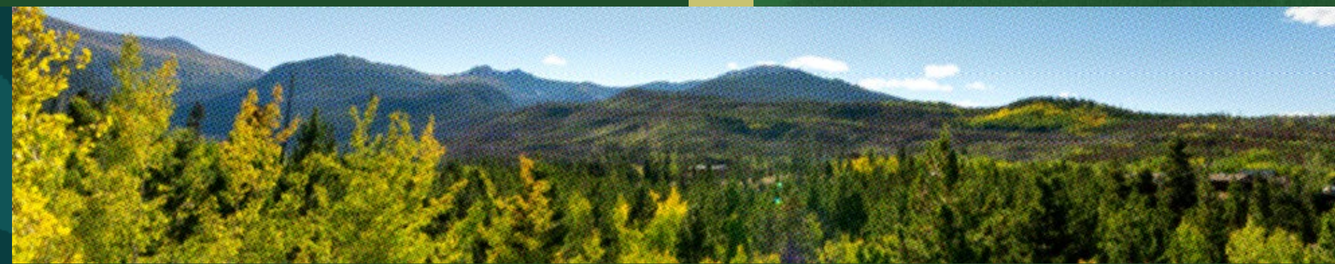
09F8021C ->
Priority=2,
PGN=129026,
DA=255,
SA=28,
Data=0E FC 24 8D A8 00 FF FF

COG = 3.61 rad (207deg)

SOG = 1.68 m/s (3.8 mph)

Sending Large Messages

CAN frames are limited to 8 bytes. What if we want to send more than 8 bytes?



NMEA Fast Packet Protocol

CAN Multiframe (First Frame)

<table>
<tr><th></th><th colspan="8">Byte 1</th><th colspan="8">Byte 2</th><th colspan="8">Byte 3</th><th colspan="8">Byte 4</th><th colspan="8">Byte 5</th><th colspan="8">Byte 6</th><th colspan="8">Byte 7</th><th colspan="8">Byte 8</th></tr>
<tr><td>CAN Bit Num</td><td>40</td><td>41</td><td>42</td><td>43</td><td>44</td><td>45</td><td>46</td><td>47</td><td>48</td><td>49</td><td>50</td><td>51</td><td>52</td><td>53</td><td>54</td><td>55</td><td>56</td><td>57</td><td>58</td><td>59</td><td>60</td><td>61</td><td>62</td><td>63</td><td>64</td><td>65</td><td>66</td><td>67</td><td>68</td><td>69</td><td>70</td><td>71</td><td>72</td><td>73</td><td>74</td><td>75</td><td>76</td><td>77</td><td>78</td><td>79</td><td>80</td><td>81</td><td>82</td><td>83</td><td>84</td><td>85</td><td>86</td><td>87</td><td>88</td><td>89</td><td>90</td><td>91</td><td>92</td><td>93</td><td>94</td><td>95</td><td>96</td><td>97</td><td>98</td><td>99</td><td>100</td><td>101</td><td>102</td><td>103</td></tr>
<tr><td rowspan="2">Fast Packet</td><td colspan="3">CP1</td><td colspan="5">CP2</td><td colspan="8">CP3</td><td rowspan="2" colspan="48">NMEA 2000 Fields per PGN (up to 6 bytes)</td></tr>
<tr><td colspan="3">Seq Num</td><td colspan="5">Frame Index</td><td colspan="8">Total Bytes for PGN</td></tr>
</table>

CAN Multiframe (Consecutive Frame)

<table>
<tr><th></th><th colspan="8">Byte 1</th><th colspan="8">Byte 2</th><th colspan="8">Byte 3</th><th colspan="8">Byte 4</th><th colspan="8">Byte 5</th><th colspan="8">Byte 6</th><th colspan="8">Byte 7</th><th colspan="8">Byte 8</th></tr>
<tr><td>CAN Bit Num</td><td>40</td><td>41</td><td>42</td><td>43</td><td>44</td><td>45</td><td>46</td><td>47</td><td>48</td><td>49</td><td>50</td><td>51</td><td>52</td><td>53</td><td>54</td><td>55</td><td>56</td><td>57</td><td>58</td><td>59</td><td>60</td><td>61</td><td>62</td><td>63</td><td>64</td><td>65</td><td>66</td><td>67</td><td>68</td><td>69</td><td>70</td><td>71</td><td>72</td><td>73</td><td>74</td><td>75</td><td>76</td><td>77</td><td>78</td><td>79</td><td>80</td><td>81</td><td>82</td><td>83</td><td>84</td><td>85</td><td>86</td><td>87</td><td>88</td><td>89</td><td>90</td><td>91</td><td>92</td><td>93</td><td>94</td><td>95</td><td>96</td><td>97</td><td>98</td><td>99</td><td>100</td><td>101</td><td>102</td><td>103</td></tr>
<tr><td rowspan="2">Fast Packet</td><td colspan="3">CP1</td><td colspan="5">CP2</td><td rowspan="2" colspan="56">NMEA 2000 Fields per PGN (to 7 bytes)</td></tr>
<tr><td colspan="3">Seq Num</td><td colspan="5">Frame Index</td></tr>
</table>

3-bit Sequence Number

Max Bytes = 31*7 + 6 = 223

```
cansend can1 18EAFFF9#14F001
```

Request Message

Trace of Fast Packet Messages

```
can1  19F01620   [8]  00 2E 02 01 02 01 2A 01   '......*.'
can1  19F01620   [8]  01 4D 61 72 65 74 72 6F   '.Maretro'
can1  19F01620   [8]  02 6E 20 31 2D 38 36 36   '.n 1-866'
can1  19F01620   [8]  03 2D 35 35 30 2D 39 31   '.-550-91'
can1  19F01620   [8]  04 30 30 20 77 77 77 2E   '.00 www.'
can1  19F01620   [8]  05 6D 61 72 65 74 72 6F   '.maretro'
can1  19F01620   [8]  06 6E 2E 63 6F 6D FF FF   '.n.com..’

can1  19F01620   [8]  20 2E 02 01 02 01 2A 01   ' .....*.'
can1  19F01620   [8]  21 4D 61 72 65 74 72 6F   '!Maretro'
can1  19F01620   [8]  22 6E 20 31 2D 38 36 36   '"n 1-866'
can1  19F01620   [8]  23 2D 35 35 30 2D 39 31   '#-550-91'
can1  19F01620   [8]  24 30 30 20 77 77 77 2E   '$00 www.'
can1  19F01620   [8]  25 6D 61 72 65 74 72 6F   '%maretro'
can1  19F01620   [8]  26 6E 2E 63 6F 6D FF FF   '&n.com..’
```

Hex:

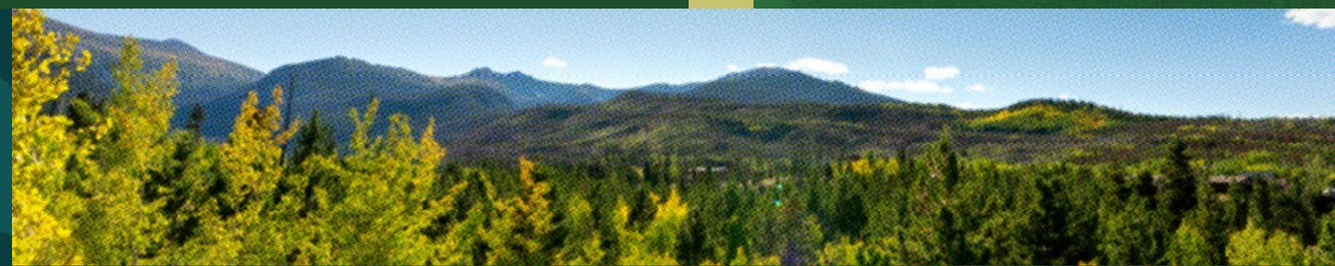
```
02 01 02 01 2A 01 4D 61 72 65 74 72 6F 6E 20 31 2D 38 36 36 35 35 30 2D 39 31
2D 30 30 20 77 77 77 2E 6D 61 72 65 74 72 6F 6E 2E 63 6F 6D
```

ASCII:

```
Maretron 1-866-550-9100 www. maretron.com
```

Address Claiming

How do we tie a source address to a device?



Interpret Address NAME Fields

- Find Industry Group
- Understand Device Class and Function Codes
- Confirm Manufacturer
- Find identifier

NATIONAL MARINE ELECTRONICS ASSOCIATION

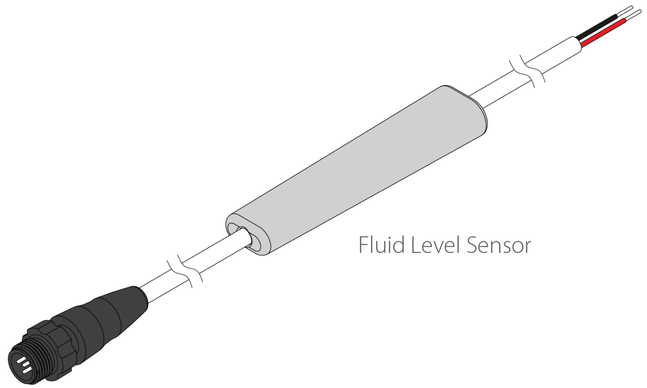
NMEA 2000 ®

Appendix B.6 -- Class & Function Codes

STANDARD FOR SERIAL-DATA NETWORKING OF MARINE ELECTRONIC DEVICES

Version 3.000
February 2022

Example Address Claim



How do we interpret the following message?

18EEFF1C [8] 21 12 80 11 00 AA A0 CF

```
 8 18EEFF1C -> Priority=6, PGN=60928, DA=255, SA=28, Data=21 12 80 11 00 AA A0 CF

ISO Address Claim

Address claimed:
```

Self-Configurable Address	Industry Group	Device Class Instance	Device Class	Function Instance	Function	ECU Instance	Manufacturer Code	Identity Number
True	4	15	80	0	170	0	140	4641

Address Claim NAME Structure (Data Field)

Field 10	Field 9	Field 8	Field 7	Field 6	Field 5	Field 4	Field 3	Field 2	Field 1
Self-Configurable Address	Industry Group	Device Class Instance	Device Class	NMEA Reserved	Function Code	Function Instance	ECU Instance	Manufacturer Code	Identity Number
1 bit	3 bits	4 bits	7 bits	1 bit	8 bits	5 bits	3 bits	11 bits	21 bits
8 bit 1	8 bit 1	8 bit 1	8 bit 1	8 bit 1	8 bit 1	8 bit 1	8 bit 1	8 bit 1	8 bit 1
Byte 8	Byte 7	Byte 6	Byte 5	Byte 4	Byte 3	Byte 2	Byte 1

Fill out the table below with data from the CAN message. Note: Bytes are reversed. Byte 1 is next to the DLC and Byte 8 is near the CRC.

Bytes	Byte 8	Byte 7	Byte 6	Byte 5	Byte 4	Byte 3	Byte 2	Byte 1
Values	CF	A0	AA	00	11	80	12	21
Nibbles	C F	A 0	A A	0 0	1 1	8 0	1 2	2 1
Bits	1100 1111	1010 0000	1010 1010	0000 0000	0001 0001	1000 0000	0001 0010	0010 0001
Hex	1 4	0x0F	0x50	0	0xAA	0	0	0x8C	0x1221
Fields	1 9	8	7	6	5	4	3	2	1
Decimal	1 4	15	80	0	170	0	0	140	4641

Address Claim Message Worksheet

Industry Group Listed in J1939 DA

```
 14 18EEFF1B -> Priority=6, PGN=60928, DA=255, SA=27, Data=21 12 80 11 00 AA A0 CF
ISO Address Claim
Address claimed:
```

Self-Configurable Address	Industry Group	Device Class Instance	Device Class	Function Code	Function Instance	ECU Instance	Manufacturer Code	Identity Number
True	4	15	80	170	0	0	140	4641

Industry Groups (IG) (Table B1)

List of Industry Group enumerations.

Return To Documentation Tab

Revised	Industry Group ID	Industry Group Description	Notes	Date Created or Last Modified
	0	Global, applies to all		10/1/1998
	1	On-Highway Equipment		10/1/1998
	2	Agricultural and Forestry Equipment		10/1/1998
	3	Construction Equipment		10/1/1998
	4	Marine		10/1/1998
	5	Industrial-Process Control-Stationary (Gen-Sets)		10/1/1998
	6	Reserved for future assignment by SAE		5/10/2000
	7	Reserved for future assignment by SAE		5/10/2000

System and Function Listed in J1939 DA

```
 14 18EEFF1B -> Priority=6, PGN=60928, DA=255, SA=27, Data=21 12 80 11 00 AA A0 CF
ISO Address Claim
Address claimed:
```

Self-Configurable Address	Industry Group	Device Class Instance	Device Class	Function Code	Function Instance	ECU Instance	Manufacturer Code	Identity Number
True	4	15	80	170	0	0	140	4641

Industry Group and Vehicle System Dependent NAME Functions (Table B12)

The NAME Functions assignments for the upper 128 Function values and the Industry Group specific Vehicle System assignments. The NAME Functions in the upper 128 NAME Function values are dependent upon the Industry Group and the Vehicle System values in NAME. Due to the dependency of these NAME Functions on Vehicle System and the dependency of Vehicle System on Industry Group, the following data defines both Vehicle System and Function for each Industry Group.The NAME fields are described in SAE J1939-81.

Return To Documentation Tab

Revised	Industry Group ID	Vehicle System ID	Vehicle System Description	Function ID	Function Description
	4	80	Instrumentation/general systems	140	Voyage Data Recorder
	4	80	Instrumentation/general systems	150	Integrated Instrumentation
	4	80	Instrumentation/general systems	160	General Purpose Displays
	4	80	Instrumentation/general systems	170	General Sensor Box
	4	80	Instrumentation/general systems	180	Weather Instruments
	4	80	Instrumentation/general systems	190	Transducer/general
	4	80	Instrumentation/general systems	200	NMEA 0183 Converter
	4	90	Environmental (HVAC) systems	255	Not Available
	4	100	Deck, cargo, and fishing equipment systems	255	Not Available

Class and Function in NMEA 2000

```
 14 18EEFF1B -> Priority=6, PGN=60928, DA=255, SA=27, Data=21 12 80 11 00 AA A0 CF
ISO Address Claim
Address claimed:
```

Self-Configurable Address	Industry Group	Device Class Instance	Device Class	Function Code	Function Instance	ECU Instance	Manufacturer Code	Identity Number
True	4	15	80	170	0	0	140	4641

Class Code	Funct Code	Class / Function Name	Description
80		Instrumentation/General Systems	Instrumentation/General systems {Deprecated class and assigned functions - see specific functions for recommendations for future use}
	170	General Sensor Box ***DEPRECATED - Function not for use in new designs!***	General sensor box {Deprecated - See Sensor Communication Interface - Class Code 75}

Determine Manufacturer from J1939 DA

```
 14 18EEFF1B -> Priority=6, PGN=60928, DA=255, SA=27, Data=21 12 80 11 00 AA A0 CF
ISO Address Claim
Address claimed:
```

Self-Configurable Address	Industry Group	Device Class Instance	Device Class	Function Code	Function Instance	ECU Instance	Manufacturer Code	Identity Number
True	4	15	80	170	0	0	140	4641

Manufacturer ID Codes (Table B10)

The list of all Manufacturer Identifier code assignments.

Return To Documentation Tab

Revised	Mfr ID	Manufacturer	Location	Date Created or Last Modified
	137	Littelfuse, Inc (formerly Maretron)	Chicago, IL USA	4/22/2023
	138	Georg Fritzmeier GmbH & Co. KG	Grosshelfendorf, Germany	9/15/2003
	139	Caterpillar Trimble Control Technologies (CTCT), LLC	Dayton, OH USA	9/15/2003
	140	Lowrance Electronics, Inc.	Tulsa, OK USA	9/18/2003
	141	Thales Navigation Ltd.	Surrey, UK	11/1/2003
	142	TRW Automotive (Commercial Steering Systems)	Lafayette, IN USA	12/20/2003
	143	W. Gmeiner GmbH & Co.	Kummersbruck, Germany	12/20/2003
	144	Mercury Marine	Fond du Lac, WI USA	12/20/2003

Address Claim Message Exercise

The following CAN message is on the NMEA2000 network.

18EEFF20 [8] 9A 65 31 11 00 91 78 C0

1. What is the source address?
2. What is the PGN?
3. What is the priority?
4. What is the destination address?
5. Fill in the table below:

Self-Configurable Address	Industry Group	Device Class Instance	Device Class	Function Instance	Function	ECU Instance	Manufacturer Code	Identity Number

6. Who is the Manufacturer?
7. What is the name of the Industry Group?
8. What is the Device Class name?
9. What is the Function name?

Address Claim NAME Structure (Data Field)

Field 10	Field 9	Field 8	Field 7	Field 6	Field 5	Field 4	Field 3	Field 2	Field 1
Self-Configurable Address	Industry Group	Device Class Instance	Device Class	NMEA Reserved	Function Code	Function Instance	ECU Instance	Manufacturer Code	Identity Number
1 bit	3 bits	4 bits	7 bits	1 bit	8 bits	5 bits	3 bits	11 bits	21 bits
8 bit 1	8 bit 1	8 bit 1	8 bit 1	8 bit 1	8 bit 1	8 bit 1	8 bit 1	8 bit 1	8 bit 1
Byte 8	Byte 7	Byte 6	Byte 5	Byte 4	Byte 3	Byte 2	Byte 1
can.message data[7]	data[6]	data[5]	data[4]	data[3]	data[2]	data[1]	data[0]

Fill out the table below with data from the CAN message. Note: Bytes are reversed. Byte 1 is next to the DLC and Byte 8 is near the CRC.

Bytes	Byte 8	Byte 7	Byte 6	Byte 5	Byte 4	Byte 3	Byte 2	Byte 1
Values								
Nibbles								
Bits								
Hex								
Decimal								

Address Claim Message Worksheet

Address Claim Exercise Answers

The following CAN message is on the NMEA2000 network.

18EEFF20 [8] 9A 65 31 11 00 91 78 C0

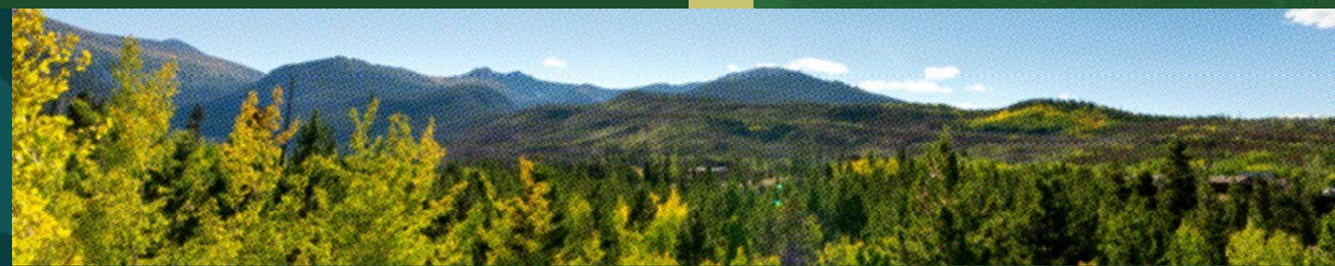
1. What is the source address? **32 (0x20)**
2. What is the PGN? **60928 (0xEE00)**
3. What is the priority? **6**
4. What is the destination address? **255 (0xFF)**
5. Fill in the table below:

Self-Configurable Address	Industry Group	Device Class Instance	Device Class	Function Instance	Function	ECU Instance	Manufacturer Code	Identity Number
True	4	0	60	0	145	0	137	1140122

6. Who is the Manufacturer? Littelfuse, Inc (formerly Maretron)
7. What is the name of the Industry Group? Marine
8. What is the Device Class name? Navigation systems
9. What is the Function name? Global Navigation Satellite System (GNSS)

Security Assessment Ideas

What are the Challenges?



CyberBoat Challenge Ideas

- Decode proprietary messages:
  - Single-Frame Proprietary:
    - PGN 61184 (0xEF00) is Destination Addressable
    - PGNs 65280 (0xFF00) through 65535 (0xFFFF)
  - Fast-Packet Proprietary
    - PGN 126720 (0x1EF00) is Destination Addressable
    - PGNs 130816 (0x1FF0) through 131071 (0x1FFFF)
- Denial of Service using Address Claims
- Explore vulnerabilities in the fast packet protocol
- Check out PGN 126208, NMEA group function
- Change the radio station on the stereo over CAN.
- Make the lights dance to music
- Set off alerts and alarms (see PGNs 126983 to 126988)
- Explore firmware updates over CAN.

Additional Resources

- CANBoat:

The End

Let’s have some fun hacking boats!