AUTOMOTIVE EMBEDDED

1. ADAS (Advanced Driver Assistance System) : It refers to a category of tech and systems designed to assist drivers in the operation of the vehicle and enhance the safety by providing features like adaptive cruise control, lane keeping assist, automotive parking and collission avoidance.

ADAS systems uses sensors such as cameras, radar, lidar, ultrasonic sensors to monitor the vehicle's surroundings and provide real time assistance to the driver.

ADAS systems includes adaptive cruise control, lane departure warning, blindspot monitoring and automatic emergency braking.

2. AUTOSAR(Automotive Open system Architecture): It is an open and standardized software architecture framework primarily focused on the development of software for Electronic control units(ECUs) within vehicles.

AUTOSAR defines a common platform, standards and specifications for automotive software development, aiming to make automotive software more modular, scalable and reusable.

AUTOSAR focuses on the overall software architecture communication, communication protocols, interfaces and development methodologies for building software systems in vehicles.

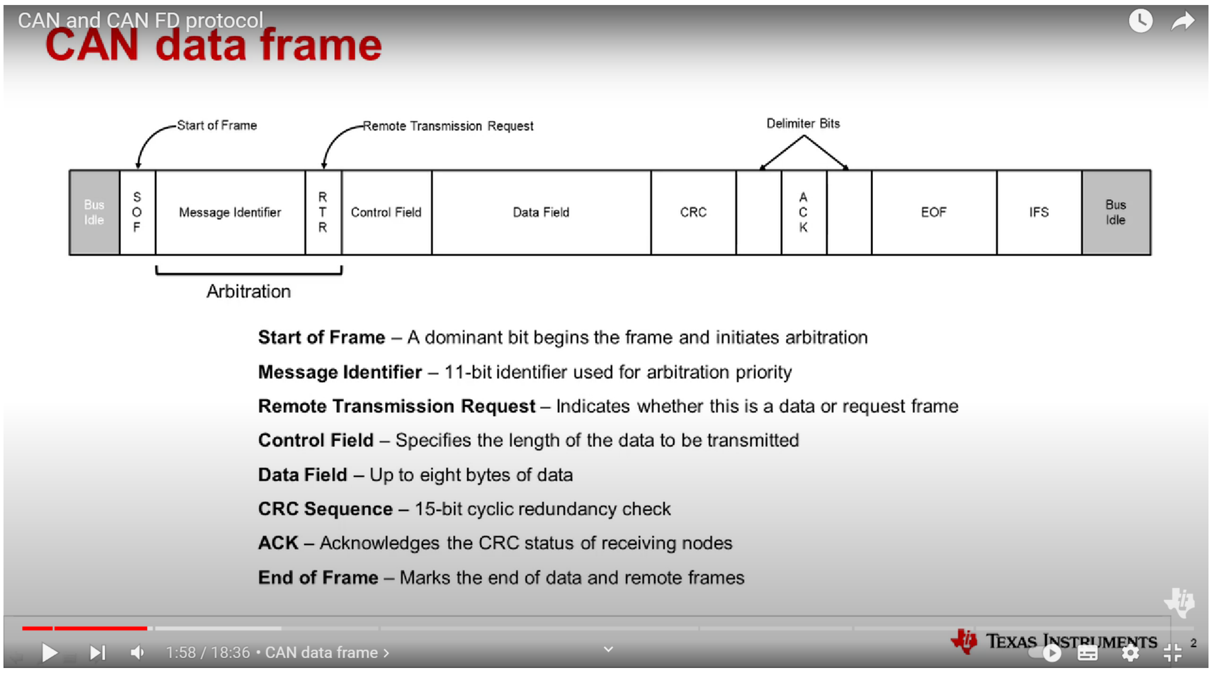
AUTOSAR is used to standardise and manage the software components responsible for functions like engine control, infotainment systems and more.

3. CAN(Controller Area Network) Networking:

It was developed to facilitate the communication between electronic control units(ECUs) in vehicles enabling them to exchange data without a central computer.

CAN typically uses a bus topology where multiple devices are connected to a single communication line. CAN high and CAN low are two bus lines kept at 3.75 V and 1.25 V.

Message frame from CAN are structured into frames, consisting of an identifier(ID), data length code(DLC), CRC(Cycle redundancy check) and Acknowledgment bits.



CAN data frame

Start of Frame – A dominant bit begins the frame and initiates arbitration

Message Identifier – 11-bit identifier used for arbitration priority

Remote Transmission Request – Indicates whether this is a data or request frame

Control Field – Specifies the length of the data to be transmitted

Data Field – Up to eight bytes of data

CRC Sequence – 15-bit cyclic redundancy check

ACK – Acknowledges the CRC status of receiving nodes

End of Frame – Marks the end of data and remote frames

7 bit IFS seperates every CAN message. SOF is always dominant 0 and is used to synchronize the bus after the idle state.

CAN uses differential signaling where the state of a bit is determined by the voltage difference between two wires(CAN-high) and (CAN-low). This provides more noise immunity

CAN uses a priority based arbitration mechanism. When two devices attempt to transmit simultaneously, the one with the lower ID wins the bus and transmits the message.

Dominant bit is logical 0 and recessive bit is logical 1. Logical 0 overrides Logical 1 in AND operation.

CAN has robust error detection and error handling mechanisms including cyclic redundancy checks(CRC) and acknowledgment mechanisms.

Data Rate : CAN supports various data rates such as 125 kbps, 250 kbps, 500 kbps and 1 Mbps, allowing flexibility in communication speed.

Extended CAN : It introduces 29 bit identifiers(instead of standard 11 bit) increasing the number of unique message IDs and enabling more complex network.

CAN FD: It is CAN with Flexible Data rate which extends the original CAN protocol by allowing variable data lengths and faster data rates. This is especially useful for high bandwidth applications.

Bit Stuffing : A bit of opposite polarity is inserted after five consecutive identical bits.

OSI(Open System Interconnection) model

CAN has a seven layer model, the CAN lower layers cover some functions of the transport layer(eg. automatic retransmission of faulty frames)

Layer 1 - Physical layer 2 - Data link layer, Layer 3 - Network layer, 4 - Transport layer, 5 - Session , Layer 6 - Presentation Layer 7 - Appllication

4. CAN Physical Layer

The CAN physical layer is a critical component of the CAN protocol, responsible for transmitting data between devices on the network. It defines how bits are physically transmitted over the communication medium, ensuring reliability and robustness.

- Differential Signaling : It uses a differential signaling scheme which means it transmits data by measuring the voltage difference between two wires.

CAN High(CAN- H) - This wire carries a voltage that is higher than the reference voltage during a dominant bit(logical 0)

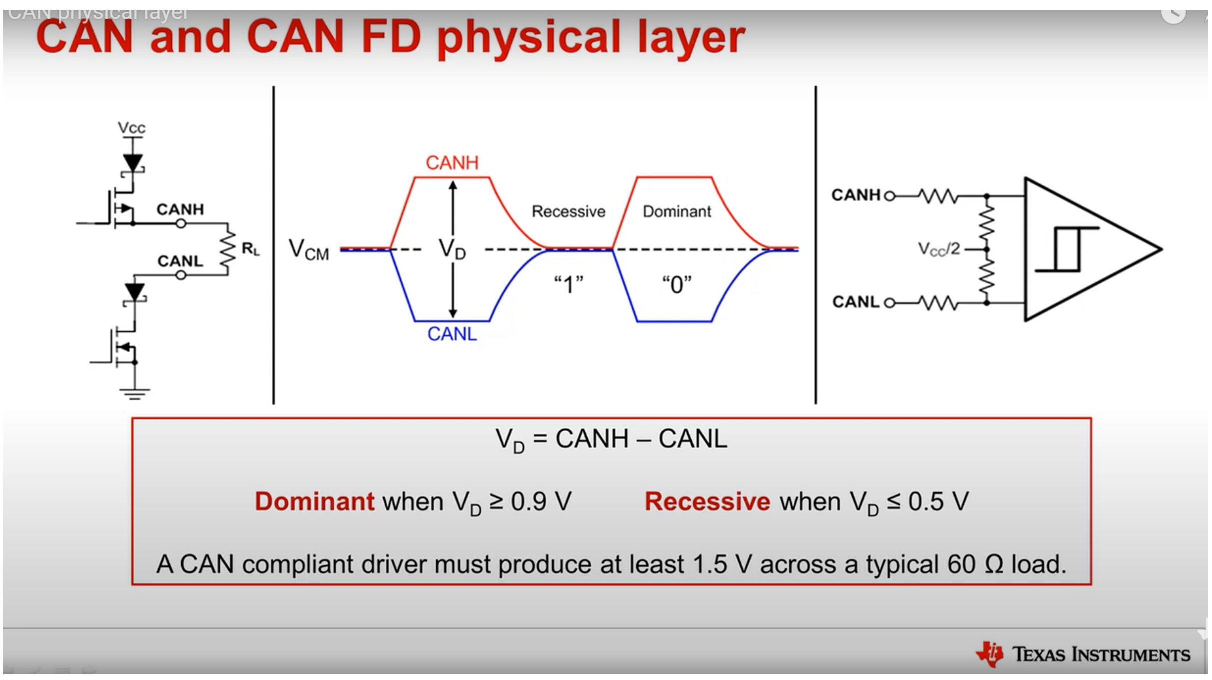
CAN Low(CAN L) : This wire carries a voltage that is lower than the reference voltage during a dominant bit(logical 0)

- Voltage levels

In differential mode, a dominant bit(0) is represented when CAN-H is greater than CAN-L whereas a recessive bit(1) is represented when CAN-H and CAN-L are approximately at same voltage levels.

Termination : Proper termination is essential in the CAN physical layer to minimize signal reflections and ensure signal integrity. Termination resistors(120 ohm) are placed at both ends of the resistors to match the characteristics impedance of the transmission line.

Bit Timing - CAN uses a bit time synchronization mechanism to ensure that all nodes on the network are in sync. It defines the length of time for each bit and the timing of the sampling points for bit reception. It is an asynchronous model.


CAN and CAN FD physical layer

V_D = CANH – CANL

Dominant when $V_D \geq 0.9$ V **Recessive** when $V_D \leq 0.5$ V

A CAN compliant driver must produce at least 1.5 V across a typical 60 Ω load.

The wiring of CAN are twisted together to reduce electromagnetic interference and form a differential signal. The wires are connected to each device on the bus through a transceiver that converts the logic level signals to the bus level signals

Connectors used in CAN networks are typically standardized such as Deutsch DT series or the AMP Superseal series.

Baud rate refers to the speed at which data is transmitted over the network, and is typically measured in bits per second(bps)

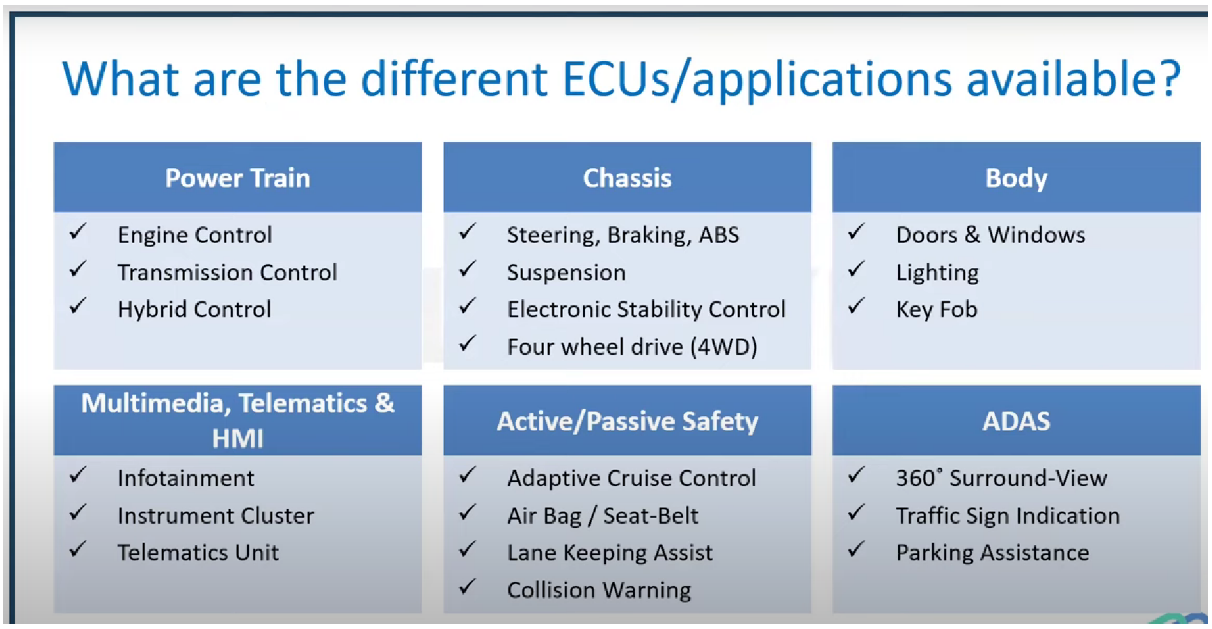
5. Introduction to Vector Tools

CANoe and CANalyzer are vector tools used for CAN network simulation, analysis and debugging.

- CANoe is a comprehensive tool that allows users to create and simulate a CAN network scenario. It provides features such as creating ECU nodes, CAPL scripting and advanced analysis capabilities. CANoe is used for simulating complex scenarios and iis suitable for intermediate to advanced users.

- CANalyzer : It is another vector tool that focuses on analyzing and debugging CAN traffic. It offers advanced logging techniques, data analysis capabilities and features like filters and triggers for in-depth analysis. It is suitable for users who require advanced logging and analysis capabilities.

6. ECUs(Electronic Control Units)



What are the different ECUs/applications available?

Power Train
- ✓ Engine Control
- ✓ Transmission Control
- ✓ Hybrid Control

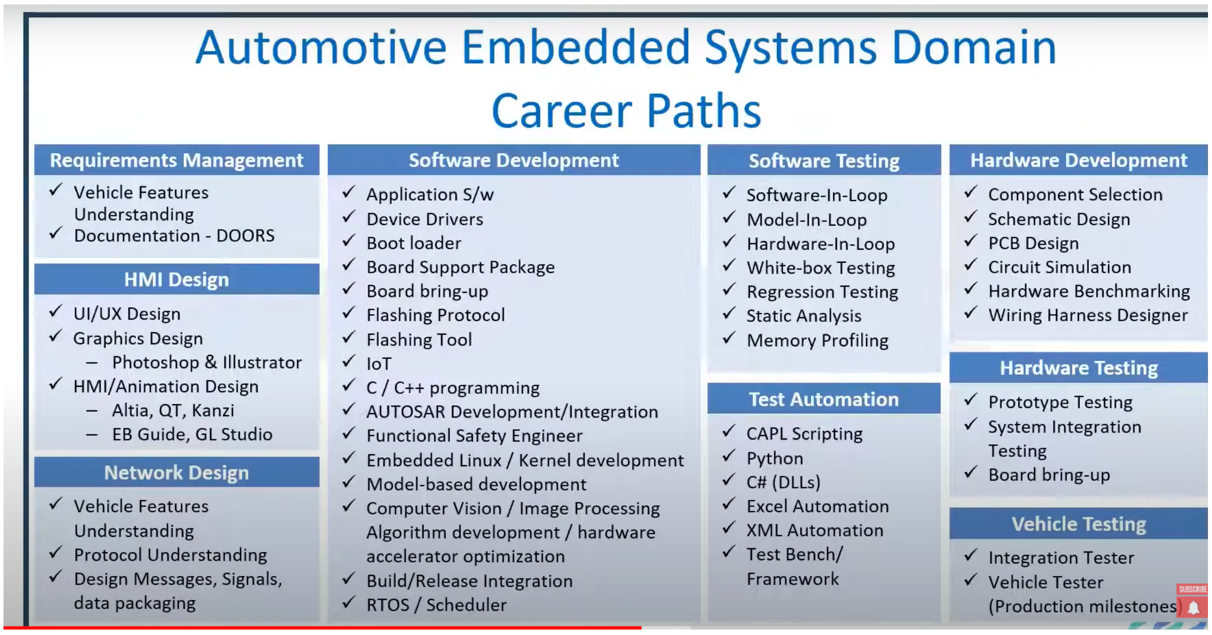
Chassis
- ✓ Steering, Braking, ABS
- ✓ Suspension
- ✓ Electronic Stability Control
- ✓ Four wheel drive (4WD)

Body
- ✓ Doors & Windows
- ✓ Lighting
- ✓ Key Fob

Multimedia, Telematics & HMI
- ✓ Infotainment
- ✓ Instrument Cluster
- ✓ Telematics Unit

Active/Passive Safety
- ✓ Adaptive Cruise Control
- ✓ Air Bag / Seat-Belt
- ✓ Lane Keeping Assist
- ✓ Collision Warning

ADAS
- ✓ 360˚ Surround-View
- ✓ Traffic Sign Indication
- ✓ Parking Assistance



Automotive Embedded Systems Domain Career Paths

Requirements Management
- ✓ Vehicle Features Understanding
- ✓ Documentation - DOORS

HMI Design
- ✓ UI/UX Design
- ✓ Graphics Design
  - – Photoshop & Illustrator
- ✓ HMI/Animation Design
  - – Altia, QT, Kanzi
  - – EB Guide, GL Studio

Network Design
- ✓ Vehicle Features Understanding
- ✓ Protocol Understanding
- ✓ Design Messages, Signals, data packaging

Software Development
- ✓ Application S/w
- ✓ Device Drivers
- ✓ Boot loader
- ✓ Board Support Package
- ✓ Board bring-up
- ✓ Flashing Protocol
- ✓ Flashing Tool
- ✓ IoT
- ✓ C / C++ programming
- ✓ AUTOSAR Development/Integration
- ✓ Functional Safety Engineer
- ✓ Embedded Linux / Kernel development
- ✓ Model-based development
- ✓ Computer Vision / Image Processing Algorithm development / hardware accelerator optimization
- ✓ Build/Release Integration
- ✓ RTOS / Scheduler

Software Testing
- ✓ Software-In-Loop
- ✓ Model-In-Loop
- ✓ Hardware-In-Loop
- ✓ White-box Testing
- ✓ Regression Testing
- ✓ Static Analysis
- ✓ Memory Profiling

Test Automation
- ✓ CAPL Scripting
- ✓ Python
- ✓ C# (DLLs)
- ✓ Excel Automation
- ✓ XML Automation
- ✓ Test Bench/ Framework

Hardware Development
- ✓ Component Selection
- ✓ Schematic Design
- ✓ PCB Design
- ✓ Circuit Simulation
- ✓ Hardware Benchmarking
- ✓ Wiring Harness Designer

Hardware Testing
- ✓ Prototype Testing
- ✓ System Integration Testing
- ✓ Board bring-up

Vehicle Testing
- ✓ Integration Tester
- ✓ Vehicle Tester (Production milestones)

7 CAN dump and CAN send

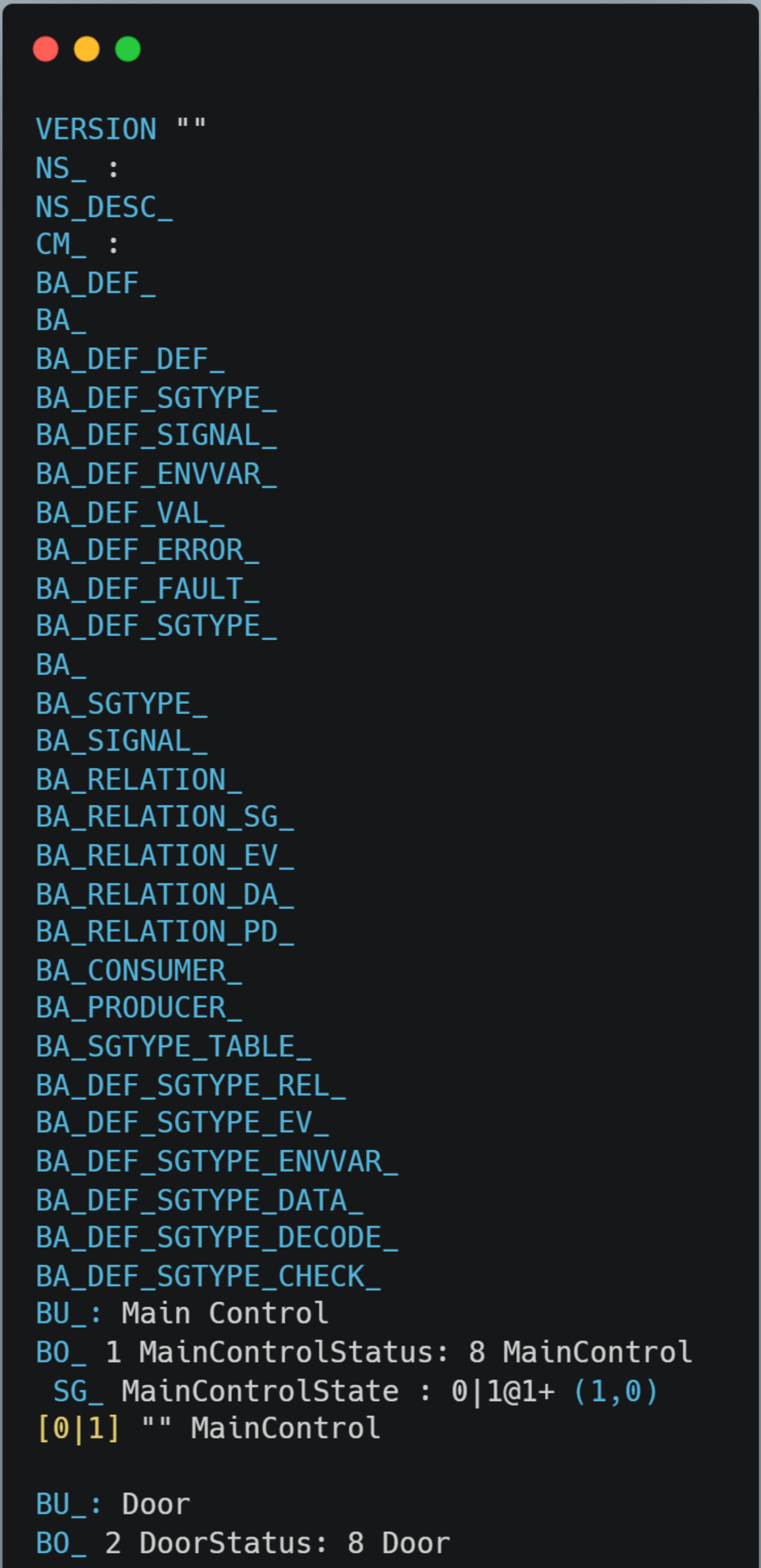
CAN dump also known as CAN logging or CAN recording, refers to the process of capturing and string the data transmitted over a CAN bus. This data can include messages from various ECUs, such as sensor readings, control commands and other information.

CAN dump is mostly used for troubleshooting, debugging and analyzing the behaviour of a CAN bus network. It allows engineers and technicians to monitor the CAN bus's data traffic to identify issues, diagnoze faults and analyze the performance of the network. It's used in automotive applications.

CAN send refers to the process of transmitting messages or data onto a CAN bus. This involes sending CAN frames with specific IDs and data payloads to other devices or ECUs on network.

CAN send is used for a range of purposes, such as controlling or configuring the ECUs, sending commands to actuators and communicating with other devices on the CAN bus. It allows for real time interaction with the CAN network, enabling features like remote control updates and reconfiguration of connected devices.

8. Common Jargons used while writing a DBC file



```
VERSION ""
NS_ :
NS_DESC_
CM_ :
BA_DEF_
BA_
BA_DEF_DEF_
BA_DEF_SGTYPE_
BA_DEF_SIGNAL_
BA_DEF_ENVVAR_
BA_DEF_VAL_
BA_DEF_ERROR_
BA_DEF_FAULT_
BA_DEF_SGTYPE_
BA_
BA_SGTYPE_
BA_SIGNAL_
BA_RELATION_
BA_RELATION_SG_
BA_RELATION_EV_
BA_RELATION_DA_
BA_RELATION_PD_
BA_CONSUMER_
BA_PRODUCER_
BA_SGTYPE_TABLE_
BA_DEF_SGTYPE_REL_
BA_DEF_SGTYPE_EV_
BA_DEF_SGTYPE_ENVVAR_
BA_DEF_SGTYPE_DATA_
BA_DEF_SGTYPE_DECODE_
BA_DEF_SGTYPE_CHECK_
BU_: Main Control
BO_ 1 MainControlStatus: 8 MainControl
 SG_ MainControlState : 0|1@1+ (1,0)
[0|1] "" MainControl

BU_: Door
BO_ 2 DoorStatus: 8 Door
```

The provided text is a snippet of a DBC (Diagnostics and Calibration Data) file, which is commonly used for defining the communication messages and signals in a CAN network.

1. `VERSION ""`: This line typically specifies the version of the DBC file format. In this case, it's left empty, which is common practice when you don't have a specific version to specify.

2. `NS_ :`: This line indicates the start of a new namespace block, which is used for organizing objects within the DBC file. The colon (:) indicates that this is an open namespace block, but it doesn't specify a namespace name.

3. `NS_DESC_`: This line is typically used to provide a description for the current namespace, but it's empty in this case, meaning there's no description provided.

4. `CM_ :`: Similar to line 2, this line indicates another namespace block, but it's left empty without specifying a namespace name**.**

5. The following lines starting with `BA_DEF_`, `BA_`, `BA_DEF_DEF_`, etc., are used to define attributes and attribute values for objects within the DBC file. These attributes can be used to provide additional information and metadata about messages, signals, nodes, etc., in the CAN network. For example, `BA_DEF_SGTYPE_` defines a signal type attribute, `BA_DEF_SIGNAL_` defines a signal attribute, and so on.

6. `BU_: Main Control`: This line defines a new ECU (Electronic Control Unit) node named "Main Control." ECU nodes represent the electronic control units in your network, and this line establishes one of them.

7. `BO_ 1 MainControlStatus: 8 MainControl`: This line defines a message (or frame) with ID 1, named "MainControlStatus," belonging to the "Main Control" ECU. The message has a data length of 8 bytes.

8. `SG_ MainControlState : 0|1@1+ (1,0) [0|1] "" MainControl`: This line defines a signal named "MainControlState" within the message "MainControlStatus." The signal has two states: 0 (Off) and 1 (On). The signal is associated with the "Main Control" ECU.

9. `BU_: Door`: Similar to line 6, this line defines another ECU node named "Door."

10. `BO_ 2 DoorStatus: 8 Door`: This line defines a message with ID 2, named "DoorStatus," belonging to the "Door" ECU. The message also has a data length of 8 bytes.

11. `SG_ DoorState : 0|1@1+ (1,0) [0|1] "" Door`: This line defines a signal named "DoorState" within the message "DoorStatus." Like the previous signal, it has two states: 0 (Close) and 1 (Open). The signal is associated with the "Door" ECU.

In summary, this DBC snippet defines two ECU nodes ("Main Control" and "Door") and messages with associated signals for each ECU. It specifies signal names, signal states, message IDs, message names, and the ECU to which each signal or message belongs within a CAN network. This information is crucial for simulating and communicating between electronic control units in a CAN network.

Alternatively, We can define a database via Vector CANdb++ and define messages inside it.

1. Define Network node which is ECU
2. Define Messages
3. Define Signal
4. Define Map signal database.
5. Edit signal with map signal database.
6. Map signal to message
7. Map network node to message.

9. Creating GUI in CANoe

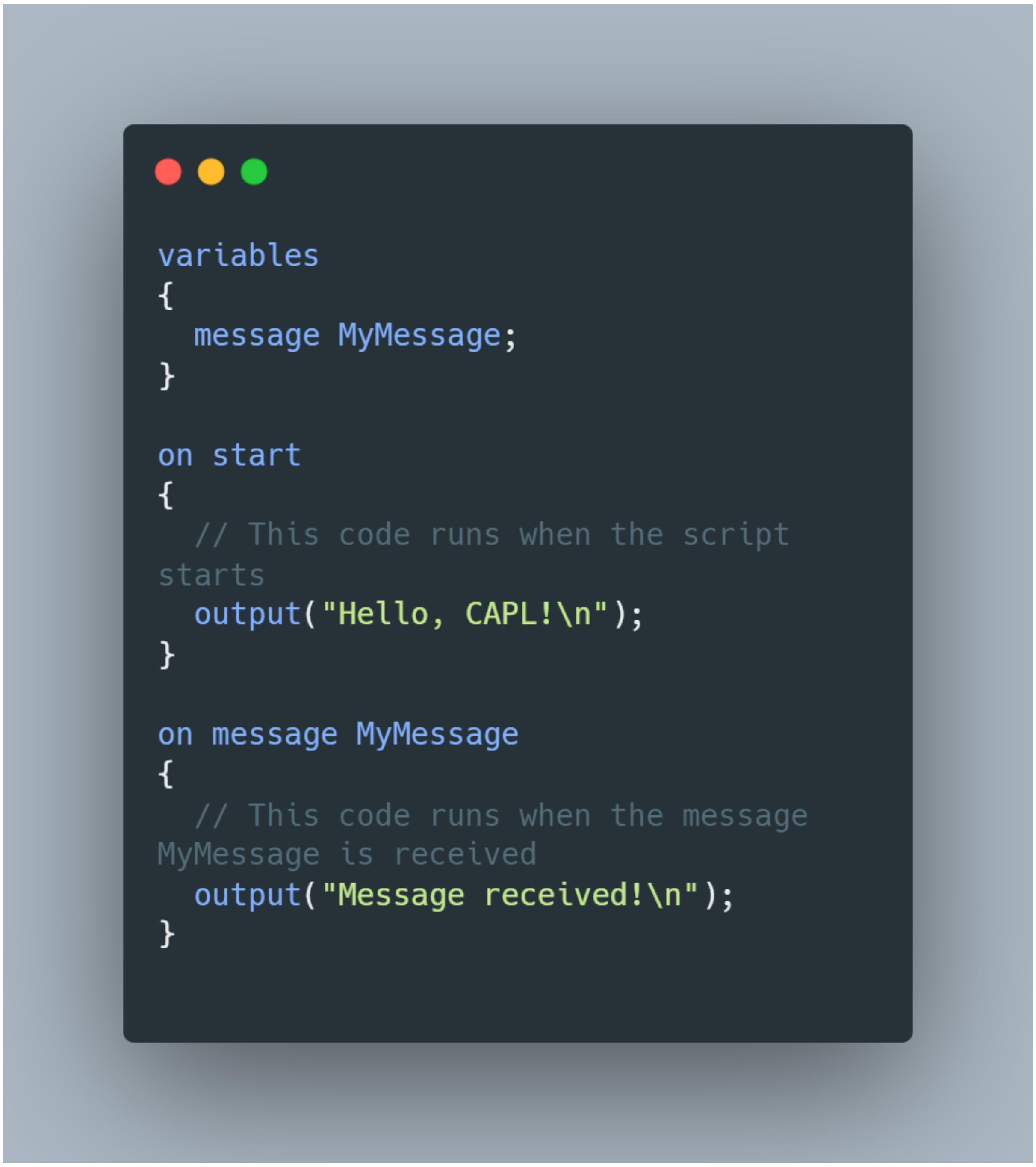
1. Go to home > Panel > create Panel.
2. Add buttons from standard control panel on the right side similar to Netbeans JDK.

10. CAPL Scripting Primer for Beginners

CAPL(Commmunication Access Programming Language) is a scripting language commonly used in the field of automotive testing and simulation. It allows you to create and control communication and simulation scenarios in various automotive software tools.

>> HELLO, CAPL !

A CAPL script consists typically of two parts : variables and functions



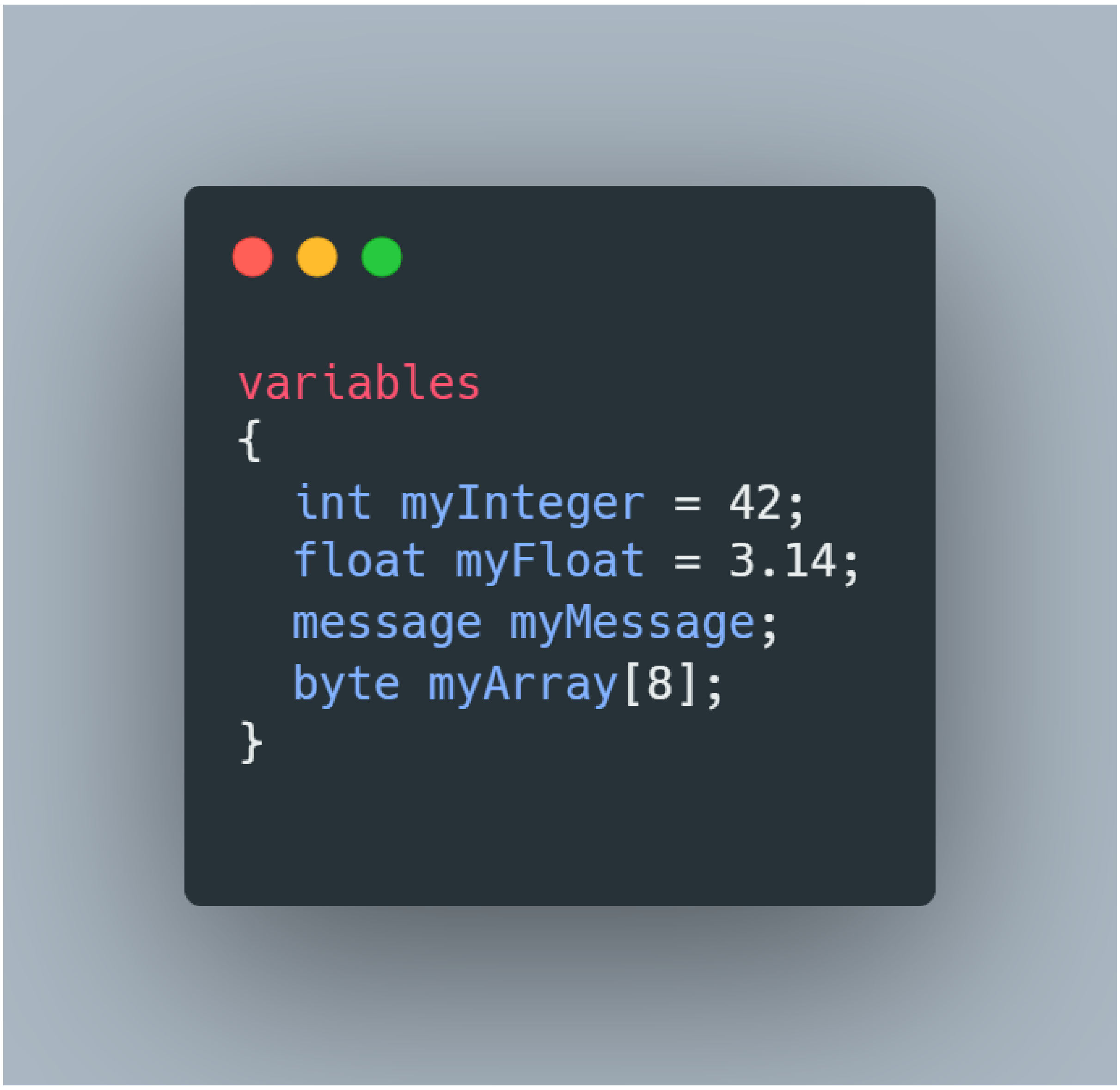
```
variables
{
  message MyMessage;
}

on start
{
  // This code runs when the script
starts
  output("Hello, CAPL!\n");
}

on message MyMessage
{
  // This code runs when the message
MyMessage is received
  output("Message received!\n");
}
```

2. Variables and Data Types

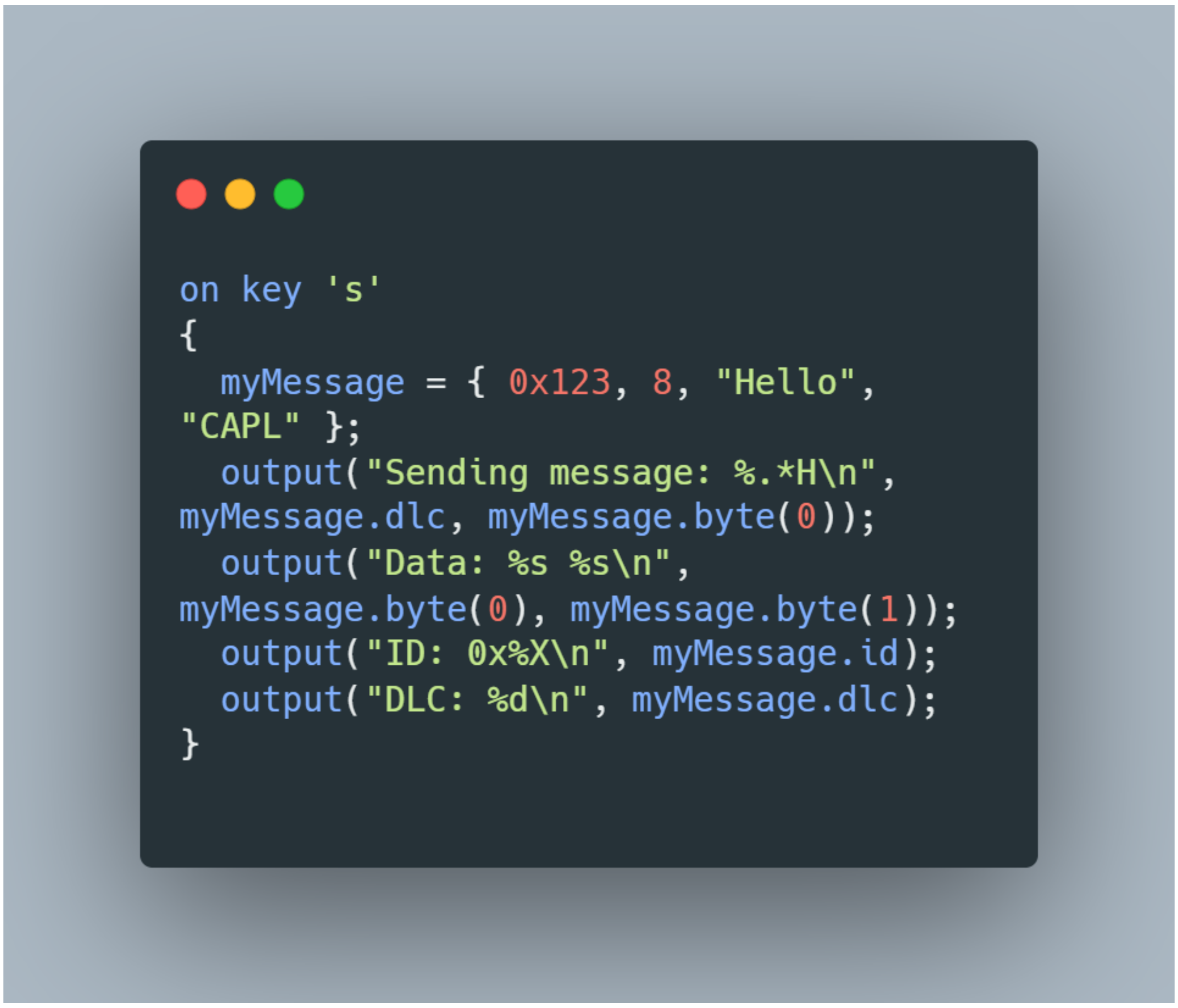
CAPL supports various data types like 'int' and 'float' and custom types like 'message'. Here's how to declare and initialise variables



```
variables
{
   int myInteger = 42;
   float myFloat = 3.14;
   message myMessage;
   byte myArray[8];
}
```

3. Sending Messages

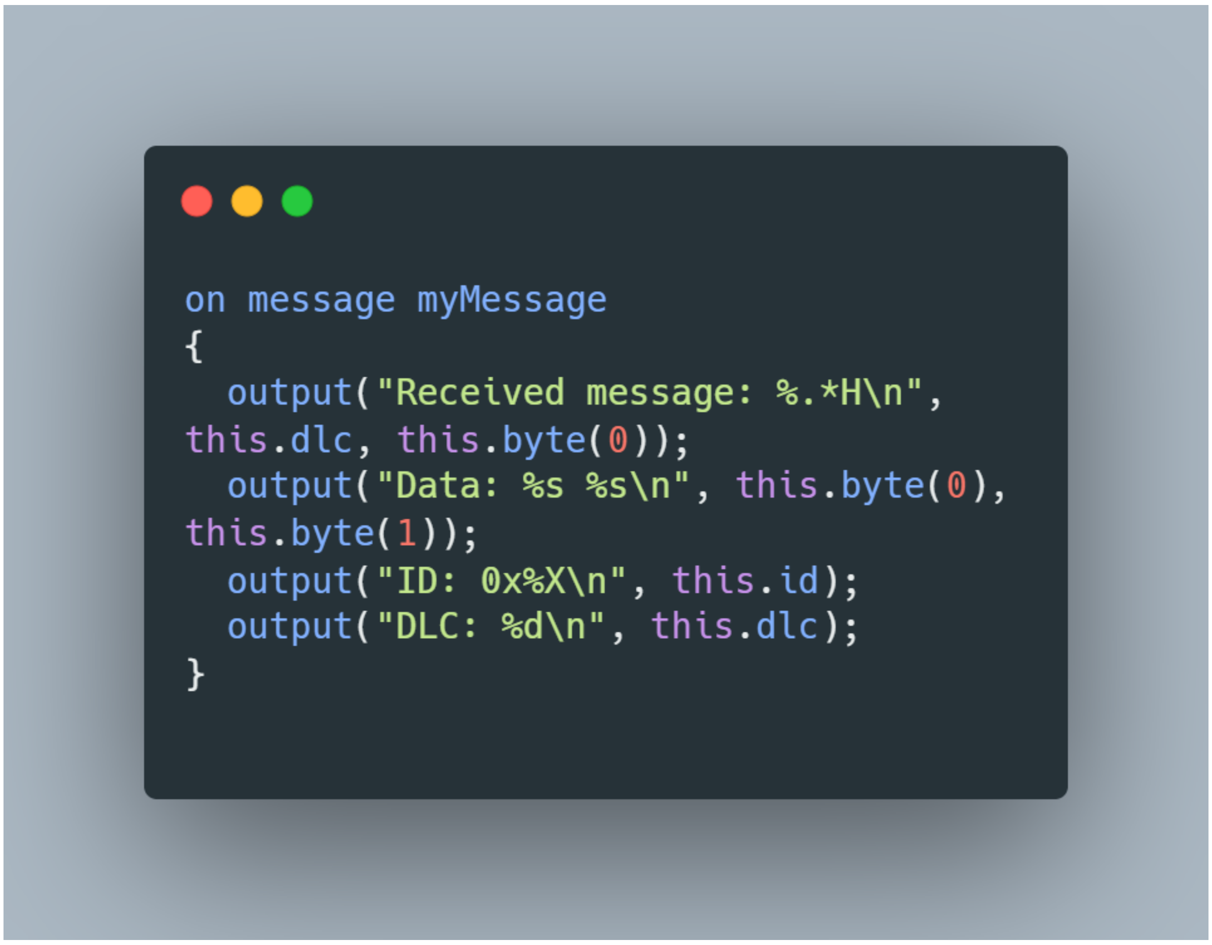
CAPL is often used to simulate messages on the CAN bus. To send a bus following snippet maybe used.



```
on key 's'
{
  myMessage = { 0x123, 8, "Hello",
"CAPL" };
  output("Sending message: %.*H\n",
myMessage.dlc, myMessage.byte(0));
  output("Data: %s %s\n",
myMessage.byte(0), myMessage.byte(1));
  output("ID: 0x%X\n", myMessage.id);
  output("DLC: %d\n", myMessage.dlc);
}
```

4. Receiving Messages

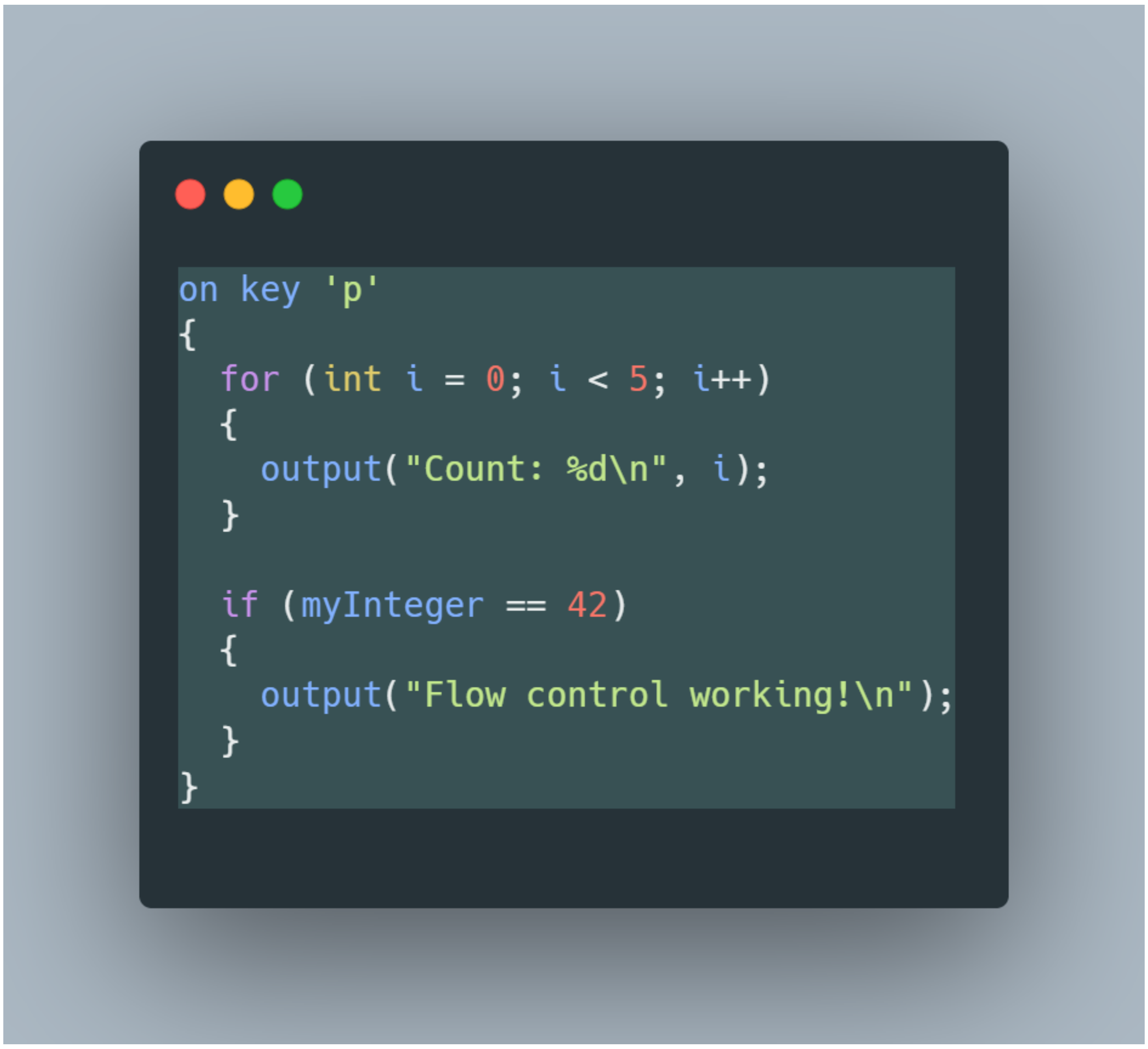
To react to incoming messages, following snippet is being used



```
on message myMessage
{
  output("Received message: %.*H\n",
this.dlc, this.byte(0));
  output("Data: %s %s\n", this.byte(0),
this.byte(1));
  output("ID: 0x%X\n", this.id);
  output("DLC: %d\n", this.dlc);
}
```

5. Flow Control

CAPL supports common programming constructs like loops and conditionals. For example



```
on key 'p'
{
  for (int i = 0; i < 5; i++)
  {
    output("Count: %d\n", i);
  }

  if (myInteger == 42)
  {
    output("Flow control working!\n");
  }
}
```

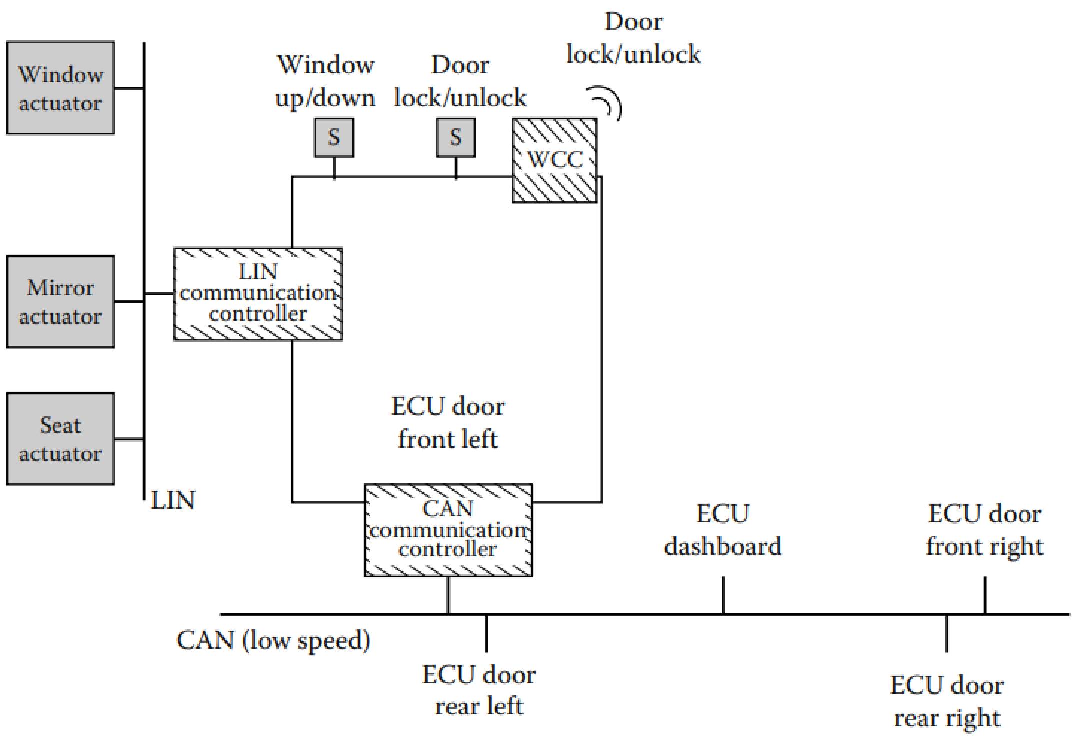
Testing the script includes plugging the CAPL script to an ECU node and compiling

11. DID YOU KNOW ?

- Today, upto 2500 signals are exchanged through up to 70 electronic control units(ECUs) on five different types of networks.
- Besides AUTOSAR, There is AEE*(Automotive Electronics Environment),EAST(Enterprise Architect for Automotive Systems), OSEK/VDEX(Open system and the corresponding interfaces for automotive electronics) are several other standards and frameworks.
- "Power train"(Control of engine and transmission) + "Chasis"(Control of suspension, braking and steering) domains are ECUs concerned with real time control and safety.
- - Principal players in the automotive industry can be divided into - Vehicle manufacturers, Automotive third part suppliers and Tool-embedded software suppliers.
- Controlling the volume of radio as per vehicle speed, When an accident causes an airbag to inflate, its microcontroller emits a signal to the embedded global positioning system receiver that then communicates with the cell phone, making it possible to give the vehicle's position to the rescue service.

- Various automotive embedded networks include LIN(Local interconnected network), CAN, TTP/C, FlexRay, MOST & IDB-1394.

- Illustrated is the examples of door/window/seat/mirror control



12. Operating Systems

- OSEK/VDX is a multitask operating system that is becoming a standard in the European automotive industry. This standard is divided into four parts - OSEK/VDX OS is the specification of the kernel itself; OSEK/VDX COM concerns the communication between tasks(internal or external ECU); OSEK/VDX NM addresses the network management and finally OSEK/VDX OIL is the language that supports the description of all the components of an application.

- OSEK/VDX OS provides services on objects like tasks("basic tasks" without blocking points and "extended tasks" that can include blocking points), events, resources ad alarms. It proposes a fixed priority(FP) schedulng policy that is applied to tasks that can be preemptive or non-preemptive and combined with a reduced version of the priority ceiling protocol. Intertask synchronization is achieved through private events and alarms.

13. Middleware

It refers to software components or layers that facilitate communication and interaction between various hardware and software components within a vehicle's electronic control system or ECU.

References

https://www.youtube.com/watch?v=mzTEZUNJFKM&ab_channel=SimplyDone

https://cdn.vector.com/cms/content/products/VectorCAST/Events/TechNights/CANalyzer_QuickStart_Apr19_Local.pdf

https://www.youtube.com/watch?v=tbhK4uA4REE&t=78s&ab_channel=AutomotiveEngineering

https://www.youtube.com/watch?v=fiMw3o4U-n0

https://youtube.com/playlist?list=PLvRkgRDxp5cqQE50te4IAMqggg-AtKbkf&si=fyl6-UB89R4AWY28

https://www.youtube.com/watch?v=8QM7oin8P4E

https://www.youtube.com/watch?v=T6xrJv9SRLI&ab_channel=EmmersiveInfotech

https://www.youtube.com/watch?v=fPjOWekzeGI&ab_channel=CARinfo3d%28En%29

https://www.youtube.com/watch?v=_Ewdksjh7zU&ab_channel=SareaHA