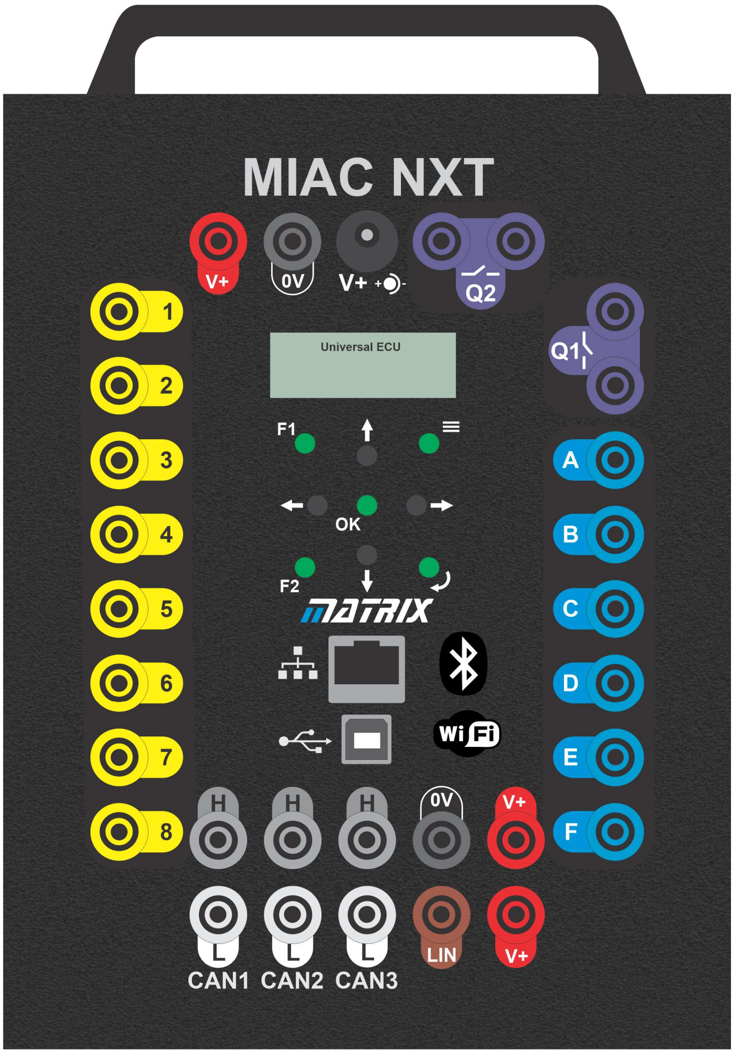
locktronics

Simplifying Electricity

CAN and LIN bus reference



MATRIX

CP5044

Contents

CAN and LIN bus reference

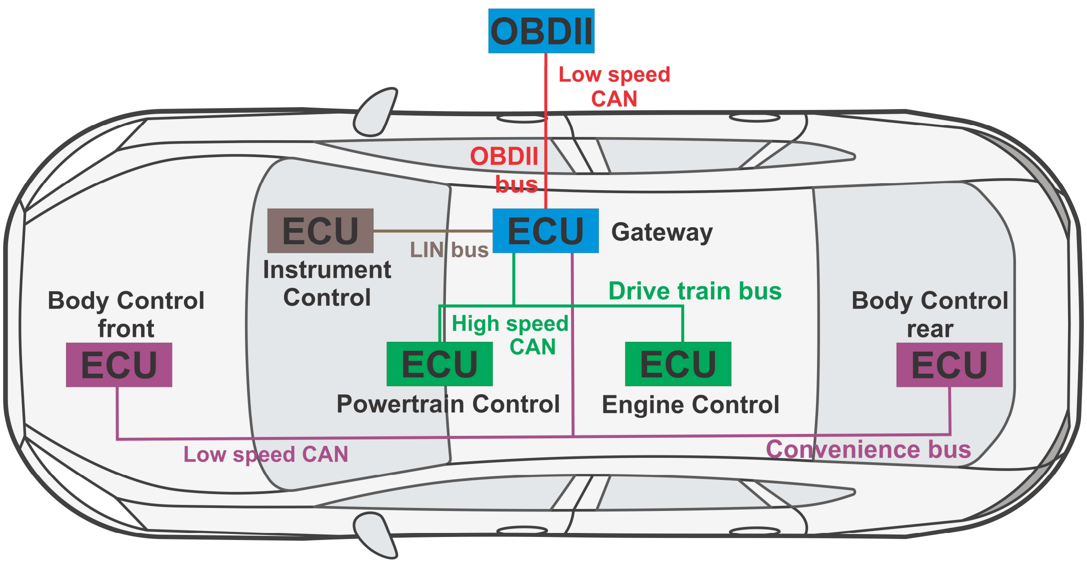
Introduction

CAN and LIN bus reference

Thirty years ago, a car was a mechanical system with some electrical components. Now, they are electronic systems with some mechanical components.

The electronic system consists of a number of ECUs (Electronic Control Units,) connected by different communications systems or 'buses'.

In this course, you will wire up a complete network of ECUs and investigate how they interconnect, how their messaging system works and how to diagnose faults in them.



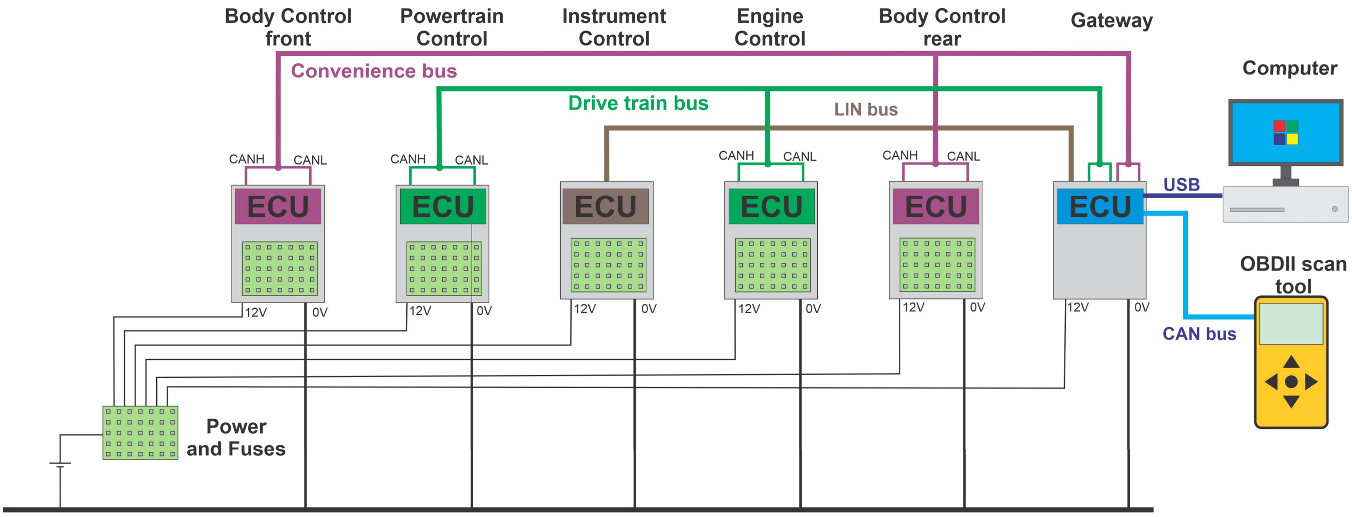
There are many different configurations for the ECUs in vehicles and several types of bus.

Our scheme uses just four different buses:

- the Drivetrain bus:
  high speed (500Kbits/s) CAN bus, connecting the Powertrain Control ECU, the Engine Control ECU and the Gateway;
- the Convenience bus:
  low speed (250kbits/s) CAN bus, connecting the front and rear Body Control ECUs and the Gateway;
- the Instrument Control bus:
  low speed LIN bus connecting the Instrument Control with the Gateway;
- the OBDII bus:
  low speed (250kbits/s) CAN bus, connecting the Gateway to the OBDII diagnostic device.

System block diagram

CAN and LIN bus reference



In a real vehicle, there would be many more ECUs and perhaps more types of bus - a Flexray bus, a MOST bus etc. This simplified six ECU system serves our purposes.

The diagram above shows how the ECUs are connected. The connections consist of the individual buses, the 12V power and 0V cables. Power is routed through a separate fuse for each node.

The LIN bus is a single wire bus with power and 0V cables.

The CAN bus is a pair of wires - CAN High and CAN Low - with power and 0V cables.

Associated with each ECU is a circuit. Initially, this consists of simple Locktronics components on a base board. As an extension, real vehicle parts can be added extending the functionality of the CAN bus.

There is a Locktronics ’Fuse panel’ with fuses and the ‘ignition’ switch.

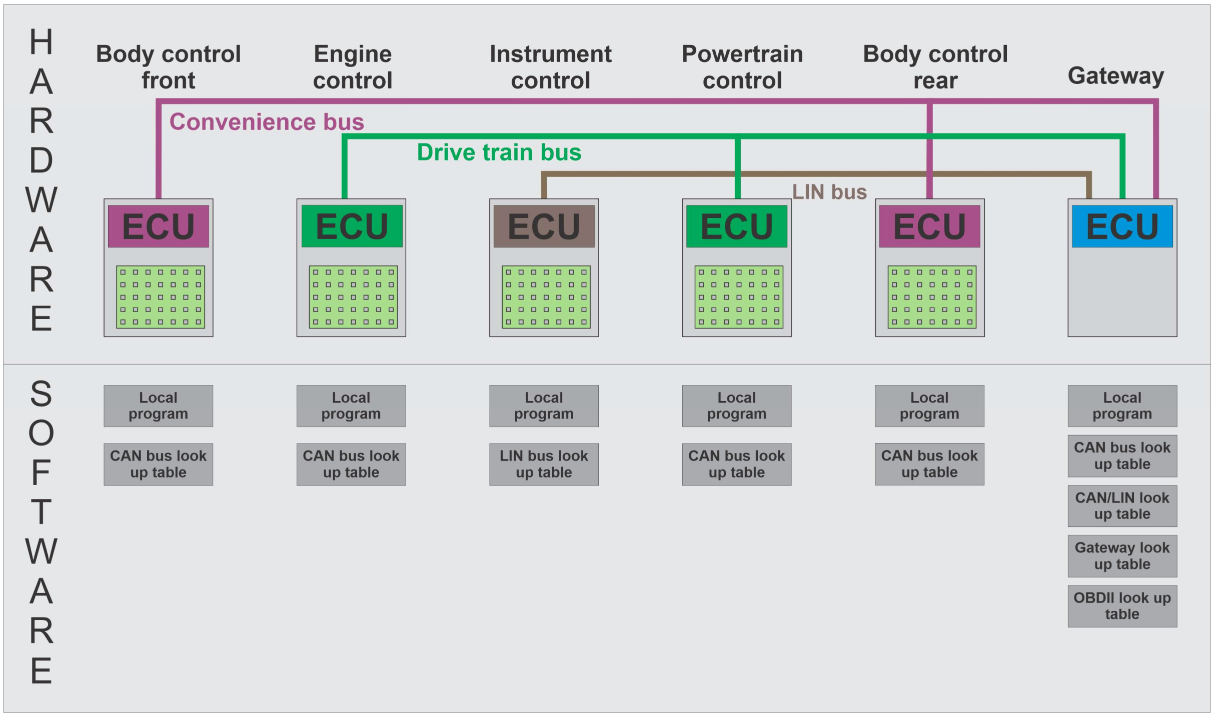
There is a PC showing instruments and diagnostic information, useful for understanding how the system works. This does not have to be connected and is for advanced learning.

Any standard OBDII scan tool can be used for fault finding.

The wiring of the complete system looks complex as there are a lot of wires needed to make up the final system. When you break it down into the individual circuit blocks with their local wiring, it is not complex.

Hardware and software

CAN and LIN bus reference



Our configuration is based on industry practice and makes sense educationally.

The diagram above identifies the different hardware and software elements in the system.

There are three main communications buses in the system:

- the Convenience CAN bus;
- the Drivetrain CAN bus;
- a single LIN bus.

There are five ECUs in the system:

- Body Control front;
- Engine Control;
- Instrument control;
- Powertrain Control;
- Body Control rear.

There is also a Gateway ECU, whose main function is to transfer messages between buses. This uses a third CAN bus to communicate with an OBDII type diagnostic tool.

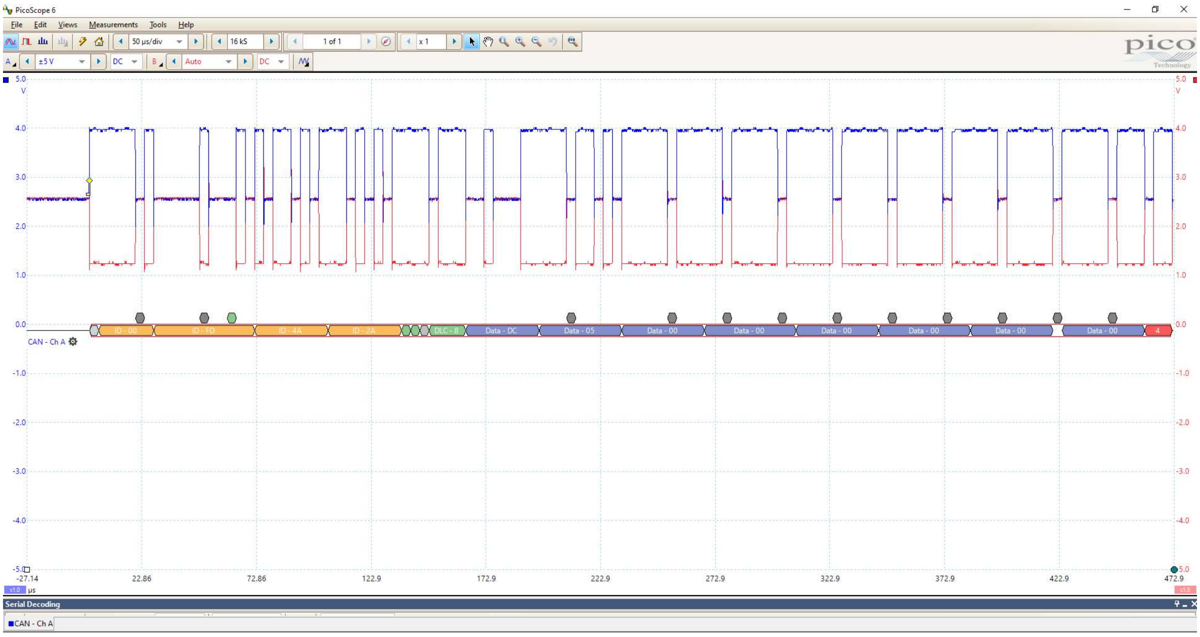
The ECUs have local circuits attached to them, consisting of a Locktronics base board and components. These represent the switches, sensors, actuators and lamps in a real system.

Each ECU has a software program. The software programs decodes the messages and adjusts the outputs accordingly. The program also makes decisions based on the state of the ECU inputs and where necessary alters local outputs or sends messages on the relevant bus, using internal look-up tables to format these messages or decode received messages.

The Gateway in this system does not have circuitry attached, but is programmed to use the CAN bus look-up table and several additional tables to decide on its functionality.

CAN bus messages

CAN and LIN bus reference



This system uses a CAN bus protocol called ‘J1939’. Unfortunately, mainstream vehicle manufacturers do not publish details of how they have implemented the CAN bus protocol. In fact, most keep the details a close secret. The advantage of J1939 is that it is an open standard and there are many documents that supports its use.

The MIAC NXT and the CAN bus solution uses J1939 at all levels of implementation - from basic microcontroller transactions upwards.

At the core of a J1939 CAN bus message is a three byte IDentifier and eight bytes of data. This is often described as: `IA IB IC D0 D1 D2 D3 D4 D5 D6 D7`

Many standard vehicle CAN messages have only two ID bytes. J1939 has three - the third being the ID of the transmitting ECU. The trace above shows a CAN message displayed on a Picoscope. CAN High (CANH) is in blue and CAN Low (CANL) is in red. The decoded message is shown below the trace.

The message here is:

```
IA IB IC D0 D1 D2 D3 D4 D5 D6 D7
FD 4A 2A DC 05 00 00 00 00 00 00
```

`IA` and `IB`, the first two bits of the identifier give the PGN - Parameter Group Number of the message. `IC` identifies the ECU that created the message - in this case node 2B.

The CAN bus signal is transmitted on two wires, with power and ground cables separate. The CANH component is transmitted between 2.5V and 4V (with respect to ground).

The CANL component is transmitted between 1V and 2.5V.

The signals are ‘differential’. At the receiver the ECU derives the voltage difference between CANH and CANL before decoding it. The image shows other parts of the message. These are start bits and error checking bits and are not covered by this course.

Binary and hex refresher

CAN and LIN bus reference

This page just reminds you how to convert between binary, hexadecimal and decimal numbers. For more information, research this on the internet.

HEX	BINARY								DECIMAL
	128	64	32	16	8	4	2	U	
01	0	0	0	0	0	0	0	1	= 1
14	0	0	0	1	0	1	0	0	= 20
55	0	1	0	1	0	1	0	1	= 85

Decimal / hexadecimal / binary conversion

Decimal	Hex	Binary
0	0	0000 0000
1	1	0000 0001
2	2	0000 0010
3	3	0000 0011
4	4	0000 0100
5	5	0000 0101
6	6	0000 0110
7	7	0000 0111
8	8	0000 1000
9	9	0000 1001
10	A	0000 1010
11	B	0000 1011
12	C	0000 1100
13	D	0000 1101
14	E	0000 1110
15	F	0000 1111
16	10	0001 0000
44	2C	0010 1100
120	78	0111 1000
255	FF	1111 1111

CAN bus look-up table

CAN and LIN bus reference

PGN DEC	PGN HEX	PG Label	Transmission Rate	SPN	SP Label
61443	F003	Electronic Engine Controller 2	45 ms	91	Accelerator Pedal Position 1
61444	F004	Electronic Engine Controller 1	40 ms	190	Engine Speed
61450	F00A	Engine Gas Flow Rate	50 ms	132	Engine Intake Air Mass Flow Rate
61463	F017	Engine Knock Level #1	5 s and on presence	1352	Engine Cylinder 1 Knock Level
61550	F06E	Fuel Pump Actuator Control	200 ms	6719	Engine Fuel pump control
64470	FBD6	Engine exhaust related parameters	500 ms	8619	Exhaust Manifold pressure
64774	FD06	Direct Lamp Control Command 2	500 ms and on change	5087	Command
64774	FD06	Direct Lamp Control Command 2	500 ms and on change	5088	Vehicle Fuel Level Low Lamp Command
64774	FD06	Direct Lamp Control Command 2	500 ms and on change	13110	Vehicle Lamp Fail Command
64775	FD07	Direct Lamp Control Command 1	500 ms and on change	5082	Engine Oil Pressure Low Lamp Command
64775	FD07	Direct Lamp Control Command 1	500 ms and on change	5083	Command
64842	FD4A	General Purpose Message #2/11	100 ms when active	1	Parking sensor
64842	FD4A	General Purpose Message #2/11	500 ms	9	Washer fluid level
64847	FD49	General Purpose Message #2/6	Startup UI	8	Indicator and Gauge Check
64848	FD50	General Purpose Message #1/9	Once on Startup	3	Gateway Who Is There Ping
64848	FD50	General Purpose Message #1/9	Reply to Gateway	4	Body Front Reply
64848	FD50	General Purpose Message #1/9	Reply to Gateway	5	Engine Reply
64848	FD50	General Purpose Message #1/9	Reply to Gateway	6	Powertrain Reply
64848	FD50	General Purpose Message #1/9	Reply to Gateway	7	Body Rear Reply
64973	FDCD	Wiper and Washer Controls	1000 ms and on change	2863	Front Operator Wiper Switch
64980	FDD4	Cab Message 3	10 s and on change	2641	Horn
64992	FDE0	Ambient Conditions 2	1 s	5581	Ambient temperature
65089	FE41	Lighting Command	1 s and on change	2367	Left Turn Signal Lights Command
65089	FE41	Lighting Command	1 s and on change	2369	Right Turn Signal Lights Command
65089	FE41	Lighting Command	1 s and on change	2375	Center Stop Light Command
65089	FE41	Lighting Command	1 s and on change	2403	Running Light Command
65089	FE41	Lighting Command	1 s and on change	2347	High Beam Head Light Command
65129	FE69	Engine Temperature 3	1 s	1636	Engine Intake Manifold 1 Temperature
65132	FE6C	Tachograph	500 ms and on change	1619	Vehicle direction
65262	FEEE	Engine Temperature	200 ms	110	Engine Coolant Temperature
65263	FEEF	Engine Fluid Level/Pressure 1	500 ms	111	Engine Coolant Level 1
65276	FEFC	Dash Display 1	1 s	96	Fuel Level 1

The table above shows all of the CAN bus codes used in the system - only a fraction of the codes used in a real vehicle - we have simplified the system to make it easier to understand.

The system is based on an open standard called 'J1939'. Car vehicle manufacturers use their own schemes for CAN and LIN bus, usually kept private. The 'J1939' protocol is used for trucks and larger vehicles. The standard is agreed by manufacturers to enable interoperability of maintenance and parts. 'J1939' is an accepted standard worldwide, open for all to use.

There are many ways to design a communications scheme. In 'J1939', messages are characterised by 'PGNs' - Parameter Group Numbers - and 'SPNs' - Suspect Parameter Numbers. The table shows you that several SPNs can be part of the same PGN.
For example PGN FE41 is the Lighting command and several SPNs are contained within it.

The PGNs are shown in hexadecimal, equivalent to decimal 65089. We work in hex because it is easier to relate to the messages on the diagnostic software.

The 'PG label' describes a messages *group* function. The 'SP label' describes its individual function. The PGN makes up the first two ID bytes of each CAN message. The SP is only a reference and is not used in the CAN message itself.

Full CAN bus look-up table

PGN DEC	PGN HEX	PG Label	Transmission Rate	SPN	SP Label	Byte(s)	Bit(s) in the byte	Bit Length	Data Mask HEX	Data Byte(s) BIN	Scaling or scheme	Offset
61443	F003	Electronic Engine Controller 2	45 ms	91	Accelerator Pedal Position 1	1	0	8	00 FF 00 00 00 00 00 00	xxxxxxxx	0.4 % per bit	0
61444	F004	Electronic Engine Controller 1	40 ms	190	Engine Speed	3	0	16	00 00 00 FF FF 00 00 00	xxxxxxxx xxxxxxxx	0.125 rpm per bit - LSB MSB	0
61450	F00A	Engine Gas Flow Rate	50 ms	132	Engine Intake Air Mass Flow Rate	2	0	16	00 00 FF FF 00 00 00 00	xxxxxxxx xxxxxxxx	0.05 kg/h per bit	0
61463	F017	Engine Knock Level #1	5 s and on presence	1352	Engine Cylinder 1 Knock Level	0	0	8	FF 00 00 00 00 00 00 00	xxxxxxxx	1 % per bit	0
61550	F06E	Fuel Pump Actuator Control	200 ms	6719	Engine Fuel pump control	0	0	16	FF FF 00 00 00 00 00 00	xxxxxxxx xxxxxxxx	0.002 5 % per bit - LSB MSB	0
64470	FBD6	Engine exhaust related parameters	500 ms	8619	Exhaust Manifold pressure	0	0	16	FF FF 00 00 00 00 00 00	xxxxxxxx xxxxxxxx	0.1 kPa per bit - LSB MSB	0
64774	FD06	Direct Lamp Control Command 2	500 ms and on change	5087	Command	0	0	2	03 00 00 00 00 00 00 00	000000xx	4 states 0 = Off, 1 = On, 2/3 = No Change	0
64774	FD06	Direct Lamp Control Command 2	500 ms and on change	5088	Vehicle Fuel Level Low Lamp Command	0	2	2	0C 00 00 00 00 00 00 00	0000xx00	4 states 0 = Off, 1 = On, 2/3 = No Change	0
64774	FD06	Direct Lamp Control Command 2	500 ms and on change	13110	Vehicle Lamp Fail Command	2	2	2	00 00 0C 00 00 00 00 00	0000xx00	4 states 0 = Off, 1 = On, 2/3 = No Change	0
64775	FD07	Direct Lamp Control Command 1	500 ms and on change	5082	Engine Oil Pressure Low Lamp Command	1	4	2	00 30 00 00 00 00 00 00	00xx0000	4 states 0 = Off, 1 = On, 2/3 = No Change	0
64775	FD07	Direct Lamp Control Command 1	500 ms and on change	5083	Command	1	6	2	00 C0 00 00 00 00 00 00	xx000000	4 states 0 = Off, 1 = On, 2/3 = No Change	0
64842	FD4A	General Purpose Message #2/11	100 ms when active	1	Parking sensor	0	0	16	FF FF 00 00 00 00 00 00	xxxxxxxx xxxxxxxx	1 mm per bit - LSB MSB	0
64842	FD4A	General Purpose Message #2/11	500 ms	9	Washer fluid level	3	0	1	00 00 00 01 00 00 00 00	0000000x	Fluid level state	0
64847	FD49	General Purpose Message #2/6	Startup UI	8	Indicator and Gauge Check	0	0	0	00 00 00 00 00 00 00 00	00000000	N/A	0
64848	FD50	General Purpose Message #1/9	Once on Startup	3	Gateway Who Is There Ping	0	0	8	FF 00 00 00 00 00 00 00	xxxxxxxx	Flag to say Gateway is present	0
64848	FD50	General Purpose Message #1/9	Reply to Gateway	4	Body Front Reply	0	0	8	00 FF 00 00 00 00 00 00	xxxxxxxx	Flag to say BodyFront is present	0
64848	FD50	General Purpose Message #1/9	Reply to Gateway	5	Engine Reply	0	0	8	00 00 FF 00 00 00 00 00	xxxxxxxx	Flag to say Engine is present	0
64848	FD50	General Purpose Message #1/9	Reply to Gateway	6	Powertrain Reply	0	0	8	00 00 00 FF 00 00 00 00	xxxxxxxx	Flag to say Powertrain is present	0
64848	FD50	General Purpose Message #1/9	Reply to Gateway	7	Body Rear Reply	0	0	8	00 00 00 00 FF 00 00 00	xxxxxxxx	Flag to say BodyRear is present	0
64973	FDCD	Wiper and Washer Controls	1000 ms and on change	2863	Front Operator Wiper Switch	0	4	4	F0 00 00 00 00 00 00 00	xxxx0000	16 states 0 = Off, 1-15 = Speed Low to High	0
64980	FDD4	Cab Message 3	10 s and on change	2641	Horn	3	2	2	00 00 00 0C 00 00 00 00	0000xx00	4 states 0 = Off, 1 = On, 2/3 = No Change	0
64992	FDE0	Ambient Conditions 2	1 s	5581	Ambient temperature	4	0	16	00 00 00 00 FF FF 00 00	xxxxxxxx xxxxxxxx	0.031 25 °C per bit - LSB MSB	-273 °C
65089	FE41	Lighting Command	1 s and on change	2367	Left Turn Signal Lights Command	1	6	2	00 C0 00 00 00 00 00 00	xx000000	4 states 0 = Off, 1 = On, 2/3 = No Change	0
65089	FE41	Lighting Command	1 s and on change	2369	Right Turn Signal Lights Command	1	4	2	00 30 00 00 00 00 00 00	00xx0000	4 states 0 = Off, 1 = On, 2/3 = No Change	0
65089	FE41	Lighting Command	1 s and on change	2375	Center Stop Light Command	2	2	2	00 00 0C 00 00 00 00 00	0000xx00	4 states 0 = Off, 1 = On, 2/3 = No Change	0
65089	FE41	Lighting Command	1 s and on change	2403	Running Light Command	0	0	2	03 00 00 00 00 00 00 00	000000xx	4 states 0 = Off, 1 = On, 2/3 = No Change	0
65089	FE41	Lighting Command	1 s and on change	2347	High Beam Head Light Command	0	6	2	C0 00 00 00 00 00 00 00	xx000000	4 states 0 = Off, 1 = On, 2/3 = No Change	0
65129	FE69	Engine Temperature 3	1 s	1636	Engine Intake Manifold 1 Temperature	0	0	16	FF FF 00 00 00 00 00 00	xxxxxxxx xxxxxxxx	0.031 25 °C per bit - LSB MSB	-273 °C
65132	FE6C	Tachograph	500 ms and on change	1619	Vehicle direction	3	6	2	00 00 00 C0 00 00 00 00	xx000000	4 states 0 = Off, 1 = On, 2/3 = No Change	0
65262	FEEE	Engine Temperature	200 ms	110	Engine Coolant Temperature	0	0	8	FF 00 00 00 00 00 00 00	xxxxxxxx	1°C per bit	-40 °C
65263	FEEF	Engine Fluid Level/Pressure 1	500 ms	111	Engine Coolant Level 1	7	0	8	00 00 00 00 00 00 00 FF	xxxxxxxx	0.4 % per bit	0
65276	FEFC	Dash Display 1	1 s	96	Fuel Level 1	2	0	8	00 FF 00 00 00 00 00 00	xxxxxxxx	0.48 L per bit	-10.5 L

CAN messages in close up

CAN and LIN bus reference

PGN HEX	PG Label	Transmission Rate	SPN	SP Label
FEEE	Engine Temperature	200ms	110	Engine Coolant Temperature

Offset Bytes	Offset Bits	Bit Length	Data Mask HEX	Data Byte(s) BIN	Scaling
0	0	8	FF 00 00 00 00 00 00 00	xxxxxxxx	1°C per bit

Let's look at a particular message and see how its constructed.

Referring to the above extract from the CAN bus look-up table, the first part of the table shows the basic message information for the engine coolant temperature command.
The Suspect Parameter number 110 information is transmitted every 200ms as part of PG group FE EE Engine Temperature.
The second part of the table shows how the information is inserted into the message:
It is 'offset' by 0 bytes, meaning it is contained in byte 0 of the eight data bytes, i.e. in D0.
The information has no 'offset bits' meaning that it is contained in bit 0 onwards.
The message has a bit length of 8 - a whole byte.
The 'Data mask' and 'Data bytes' summarise visually the offset bytes, bits, and bit length .
The scaling is 1^0C per bit - a straightforward number.

Here is an example of a CAN message with this information:

```
IA IB IC D0 D1 D2 D3 D4 D5 D6 D7
FE EE 2C 22 00 00 00 00 00 00 00
```

`IA` and `IB` provide the PGN: FE EE. `IC` is the unique ECU ID transmitting the message - in this case 2C which is our Powertrain ECU.
D0 contains the temperature in hexadecimal, 22, which is 34 decimal.
In other words, the Powertrain ECU is saying that the engine temperature is 34^0C.

CAN messages in close up

CAN and LIN bus reference

PGN HEX	PG Label	Transmission Rate	SPN	SP Label
FE41	Lighting Command	1 s and on change	2375	Center Stop Light Command

Offset Bytes	Offset Bits	Bit Length	Data Mask HEX	Data Byte(s) BIN	Scaling
2	2	2	00 00 0C 00 00 00 00 00	0000xx00	4 states 0 = Off, 1 = On, 2/3 = No Change

The 'Stop light' command is found in parameter group FE 41, 'Lighting Command', Suspect Parameter Number 2375, 'Center Stop light' command.

The offset is two bytes and so the information is in data byte D2.

The information has two offset bits in byte D2, meaning that it is found in bits 2 and 3 inside byte D2. (Remember that bit 0 is on the right.) In these two bits 00 is off, 01 is on, 10 and 11 have no effect.

Here is an example of a CAN message with this information:

```
IA IB IC D0 D1 D2 D3 D4 D5 D6 D7
FE 41 2C 00 00 04 00 00 00 00 00
```

`IA` and `IB` are the PGN: FE 41. `IC` is the unique ECU ID transmitting the message - in this case 2C, our Powertrain ECU.

Byte D2 contains the information 0000 0010 binary or 04 hex.

The message is 01, or 'on', meaning that the Powertrain is saying that the stop light should be on.

A bit of logic

CAN and LIN bus reference

The arrangement for sending digital information seems complex.

The 'J1939' standard uses two bits to describe an on/off value rather than just a single bit.

The specification specifies four states '0' = Off, '1' = On, '2 and 3' = No Change

We could just use '0' = Off, '1' = On. Why the complication?

The reason is that a single SPN could be used by more than one ECU and that we need to allow for this.

For example, consider a situation where a single SPN is used to trigger four separate warning lights, controlled by several ECUs, using just one byte in a PGN. The system must allow each ECU to alter the state of a single light at a time without affecting the state of the other lights.

PGN DEC	PGN HEX	PG Label	Transmission Rate	SPN	SP Label
6000	1770	Warning lamps	1 s and on change	100	warning A
6000	1770	Warning lamps	1 s and on change	101	warning B
6000	1770	Warning lamps	1 s and on change	102	warning C
6000	1770	Warning lamps	1 s and on change	103	warning D

Byte(s)	Bit(s) in the byte	Bit Length	Data Mask HEX	Data Byte(s) BIN	Scaling or scheme
4	6	2	00 00 00 00 FF 00 00 00	xx000000	4 states 0 = Off, 1 = On, 2/3 = No Change
4	4	2	00 00 00 00 FF 00 00 00	00xx0000	4 states 0 = Off, 1 = On, 2/3 = No Change
4	2	2	00 00 00 00 FF 00 00 00	0000xx00	4 states 0 = Off, 1 = On, 2/3 = No Change
4	0	2	00 00 00 00 FF 00 00 00	000000xx	4 states 0 = Off, 1 = On, 2/3 = No Change

The same byte in the same PGN is used by several ECUs.

If the ECU uses two bits for a logical state transmission, then it can alter the value of any warning lamp and yet leave the others unchanged.

For example:

- a transmission of 11001111 says:
  warning A: no change
  warning B: off
  warning C: no change
  warning D: no change
- a transmission of 11011111 says:
  warning A: no change
  warning B: on
  warning C: no change
  warning D: no change

LIN bus look-up table

CAN and LIN bus reference

LIN bus look up table									
LIN ID	ID Label	Data Label	Transmission Rate	Byte(s)	Bit(s) in the byte	Bit Length	Data Mask HEX	Data Byte(s) BIN	Scaling or scheme
1	Switch Poll Request	Flag Wiper %	80ms	1	0	8	FF 00 00 00 00 00 00 00	xxxxxxxx	0.4% per bit
1	Switch Poll Request	Lamp Low	80ms	1	0	1	00 01 00 00 00 00 00 00	0000000x	On or Off
1	Switch Poll Request	Lamp High	80ms	1	0	1	00 00 01 00 00 00 00 00	0000000x	On or Off
1	Switch Poll Request	Lamp Left	80ms	1	0	1	00 00 00 01 00 00 00 00	0000000x	On or Off
1	Switch Poll Request	Lamp Right	80ms	1	0	1	00 00 00 00 01 00 00 00	0000000x	On or Off
1	Switch Poll Request	Flag Horn	80ms	1	0	1	00 00 00 00 00 01 00 00	0000000x	On or Off
1	Switch Poll Request	Alive (AA)	80ms	1	0	8	00 00 00 00 00 00 FF 00	xxxxxxxx	
1	Switch Poll Request	Alive (55)	80ms	1	0	8	00 00 00 00 00 00 00 FF	xxxxxxxx	
2	Lamp Status Send	Lamp Fuel	350ms	1	0	1	01 00 00 00 00 00 00 00	0000000x	On or Off
2	Lamp Status Send	Fault Count	350ms	1	0	8	00 FF 00 00 00 00 00 00	xxxxxxxx	>0 = Lamp On

There is only one LIN bus in our system, though in a practical vehicle there would probably be several. The table above shows a summary of the messages in the LIN bus.

They are simpler than CAN bus messages and are designated as either 'Transmit (TX)' or 'Receive (RX)' messages.

Gateway look-up table

CAN and LIN bus reference

Gateway look up table			
PGN	**Origin**	**Destination**	**Function**
F004	Drive	OBDII	Engine Speed Stat
F06E	Drive	Convenience	Fuel Pump Control
FD06	Convenience	OBDII	Brake Light Fault C1223
FE41	Drive	Convenience	Lighting Command
FE6C	Drive	Convenience	Tachograph
FEEE	Drive	OBDII	Engine Coolant Temp Stat
FEFC	Convenience	OBDII	Fuel Level Circuit Error P0090
FDE0	Convenience	Drive	Ambient temperatue

The Gateway ECU performs a function as an ECU and also as a communications hub for the different buses on the system and between the system and the diagnostic software.

A key function of the Gateway is to transfer messages from one bus to another. Some messages, for example engine speed, are needed by ECUs on more than one bus.

The operation of the Gateway is defined by a look-up table, shown above, that lists the message PGNs that are transferred between buses.

OBDII code look-up table

CAN and LIN bus reference

Diagnostic codes look up table			
Description	**actual fault**	**OBD 2 dtc**	**Description**
Faulty stop light (both)	no current or no feedback	C1223	Lamp Brake Warning Output Circuit Failure
Fuel level sensor high	short to 12V	P0463	Fuel level circuit high error
Fuel level sensor low	short to 0V	P0462	Fuel level circuit low error
Low battery voltage low	below a threshold (50% / 6V)	B1318	Battery Voltage Low
CAN ECU error Front	Convenience CAN bus error	U0141	Lost Communication Body Control Front
CAN ECU error Engine	Drive CAN bus error	U0142	Lost Communication Engine Control
CAN ECU error Instrument	LIN bus error	U0143	Lost Communication Instrument Control
CAN ECU error Powertrain	Drive CAN bus error	U0144	Lost Communication Powertrain Control
CAN ECU error Rear	Convenience CAN bus error	U0145	Lost Communication Body Control Rear

A key function of the Gateway is to transfer errors to the OBDII diagnostic tool.

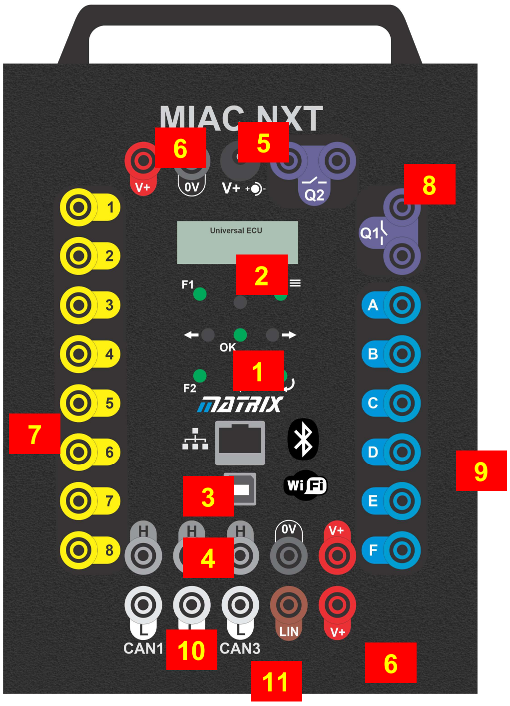
The look-up table above lists these errors - as you can see there are not that many.

When an error occurs in any of the ECUs, the error message is sent to the Gateway, where it is stored. When an OBDII tool is connected to the car, it has an option to interrogate the Gateway to find any error codes. These are then displayed on the OBDII device.

The OBD II device has the ability to clear any error codes,.

MIAC NXT introduction

CAN and LIN bus reference



1. Keypad
2. LCD display
3. RJ45 connector
4. USB connector
5. 12V 2.1mm positive inner connector
6. Power/ ground connectors
7. Eight digital or analogue inputs
8. Two relay-controlled outputs
9. Six transistor-controlled PWM outputs
10. Three CAN bus connectors
11. LIN bus connector

'MIAC' stands for Matrix Industrial Automotive Controller. 'NXT' stands for NeXT generation.

MIAC NXT is designed for educational purposes, allowing students to experiment with various types of control system.

Each MIAC has eight analogue or digital inputs, two relay-controlled outputs, and six transistor-controlled outputs.

It can communicate with other automotive applications via three CAN bus interfaces or its LIN bus interface.

The unit has a USB interface, RJ45 Ethernet interface, internal Bluetooth and internal Wi-fi modules.

Inputs are fed into a signal-conditioning circuit which allows them to be used as either analogue and digital inputs, dictated by the software. They are not optically isolated. The input range of 0 to 12V DC makes the MIAC compatible with industry standard sensors.

Two outputs from the internal PIC processor are fed into power stages, giving current amplification for four separate relays.

Relay contacts are not current-limited and so external fuses should be used to limit relay current to 8A AC or DC.

Six additional outputs are fed into a driver stage, which includes current monitors to limit output current and protect the motor driver chip in the event of short-circuits.

The internal processor connects to three CAN bus driver circuits and a LIN bus circuit to allow a number of MIACs to be linked to form a control network.

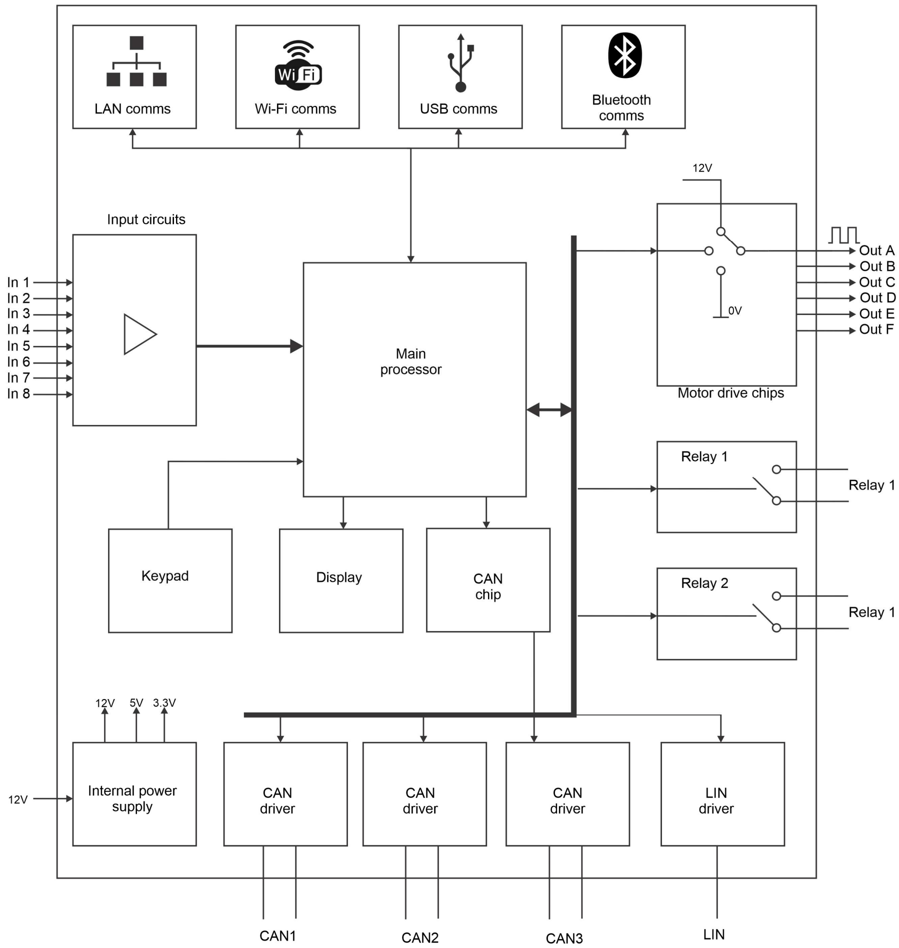
The MIAC is electrically rugged. Any output can be short-circuited to any input or any other output without the unit failing.

Control and monitoring of processes is facilitated by a four-line LCD display and a keypad.

MIAC NXT - Matrix part number: MI5550

MIAC NXT block diagram

CAN and LIN bus reference



Internally, the MIAC is powered by a powerful 24 series PICmicro device which connects directly to the USB port for fast programming and USB communications.

The PIC device is pre-programmed with a bootloader program and a Windows utility which allows programmers to download PIC compatible hex code into the device.

The PIC processor includes two internal CAN bus driver circuits. These are fed to external CAN line driver circuits for CAN buses 1 and 2. An additional CAN driver chip and line driver is included to form the third CAN bus. (Three CAN buses are needed for some applications.) A simple LIN line driver circuit is included for LIN bus communications.

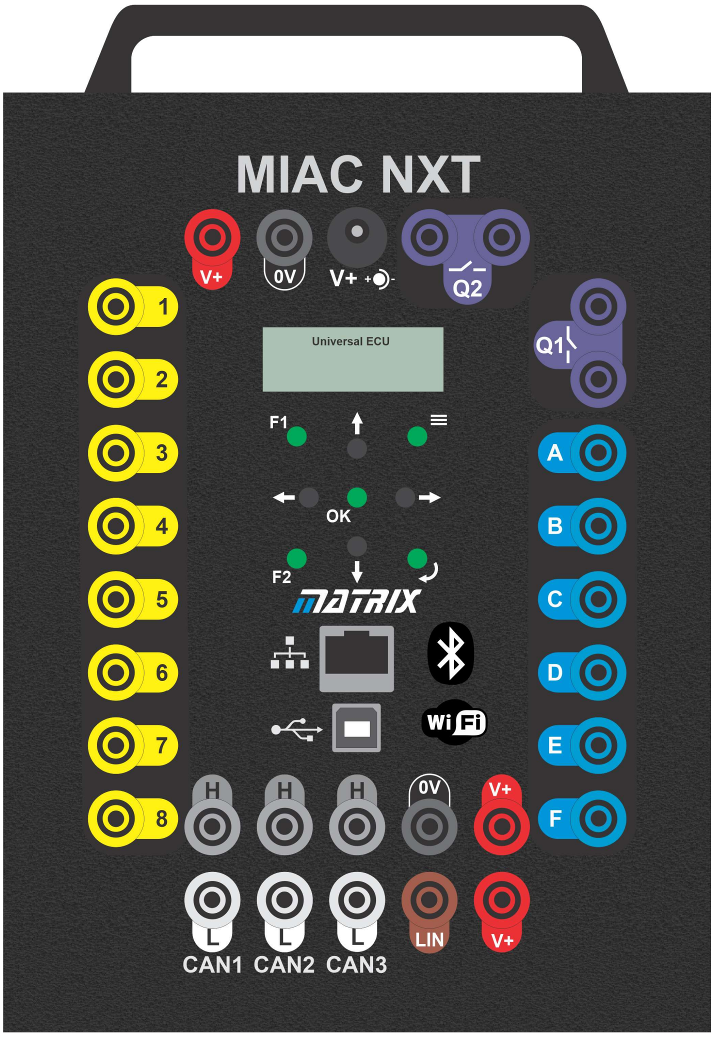
MIAC can be powered with a DC supply voltage in the range 12V to 24V DC. This can be supplied via the 2.1mm power jack (POWER), or the power supply terminals (V+, 0V) which are wired in parallel with the 2.1mm power jack. Internal power supply circuitry provides 12V, 5V and 3.3V power rails to all parts of the MIAC.

The PIC24 includes USB circuitry to provide USB connection for reprogramming and communications. Internally Bluetooth and Wi-fi modules provide communications for control, data transfer and reprogramming.

For further details on the MIAC please see the MI3728 datasheet.

MIAC NXT operating instructions

CAN and LIN bus reference



For the Sense and Control and CAN bus learning packages, the MIAC must be loaded with LK7638 firmware.

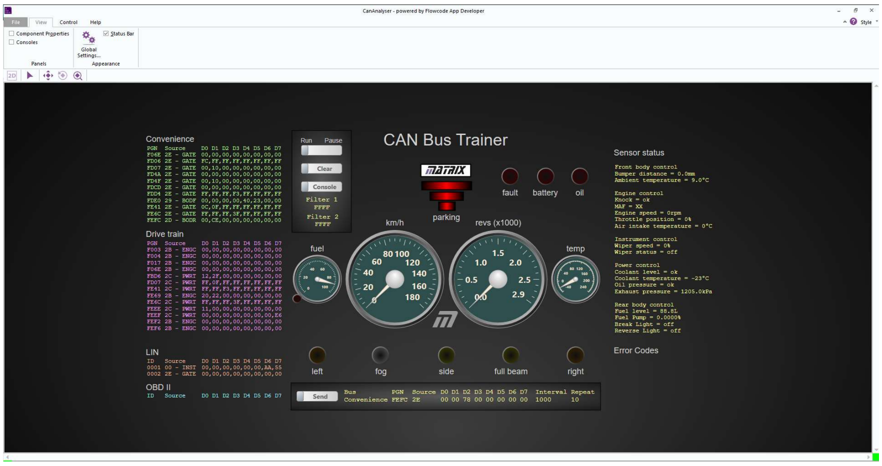
The LK7638 program contains the programs for two learning packages: Sense and Control II and CAN bus II. It allows users to control the MIAC for each worksheet in these packages.

The menu system and keypad allows you to choose the package.

When CAN bus is chosen then the menu system and keypad allow you to choose which node program the MIAC runs.

MIAC NXT diagnostic software set up

CAN and LIN bus reference



The EFIS diagnostic software shows you what is going on in the CAN/LIN bus system.

MIAC is a reprogrammable rugged dsPIC microcontroller. It uses a HID USB system so that no drivers are needed to connect it to your PC. (MIAC works only with Windows PCs.)

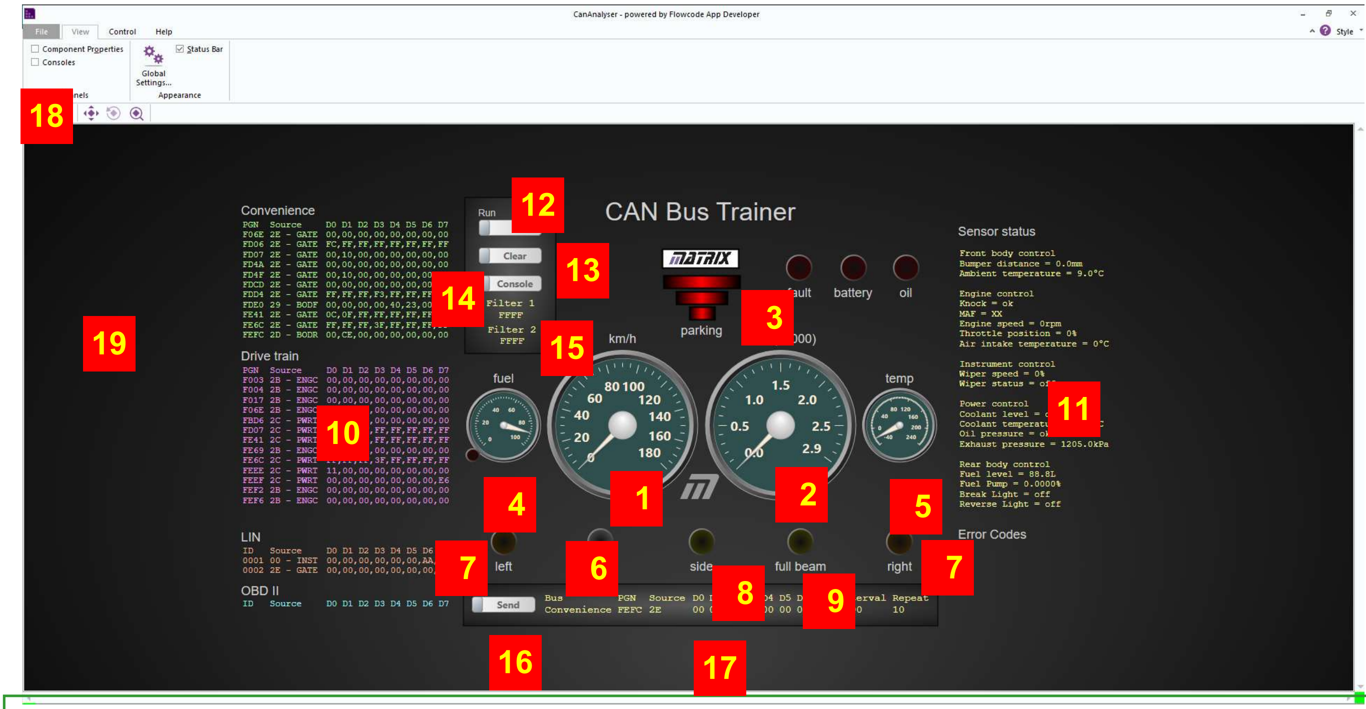
To get the MIAC working with the EFIS system, two steps are needed:

1. The MIAC needs programming with the Allcode Application Programming Interface. This API provides a number of standard functions for the PC program that can be called via USB, Wifi, or Bluetooth.
   With product code MI8975, it is available from the Matrix web site. If you have bought a CAN bus system, this will be pre-installed.
2. Your PC needs the Flowcode App Developer firmware program, which is used to show CAN bus traffic on the EFIS system, as shown above left.

The firmware product code is LK7638. It is available from the Matrix web site as a free download. You can program the MIAC with this firmware using the ‘Mloader’ utility, also available as a free download from the Matrix web site.

Diagnostic software introduction

CAN and LIN bus reference



The diagnostic software shows what is happening in the CAN/LIN bus system. It is partly diagnostic tool and partly visual representation of the CAN bus and appropriate parameters. The instrument console in modern cars is simply a computer with a similar Human Machine Interface (HMI) on a custom-made graphical display.

The items displayed on the software are as follows:

1. Speed dial;
2. Rev counter;
3. Distance marker for the parking sensor on Body Control front;
4. Fuel level with warning light;
5. Coolant temperature;
6. Fault light;
7. Left and right indicator lights which flash when indicating;
8. Side light indicator;
9. Full beam indicator;
10. CAN / LIN / OBDII bus display field;
11. Sensor status field;
12. Pause / Run control - pauses the collection of data;
13. Clear button - clears display fields;
14. Console button - brings up additional console with time stamped data;
15. Console filter - only gathers data with the ‘Filtered PGN’;
16. Send button - sends the CAN bus message that you set up in the text fields;
17. Send text fields - click on these and enter the data that you want to send;
18. GO button: press play to start the App;
19. Background - right-mouse click to change the origin and zoom.

Diagnostic software close up

CAN and LIN bus reference

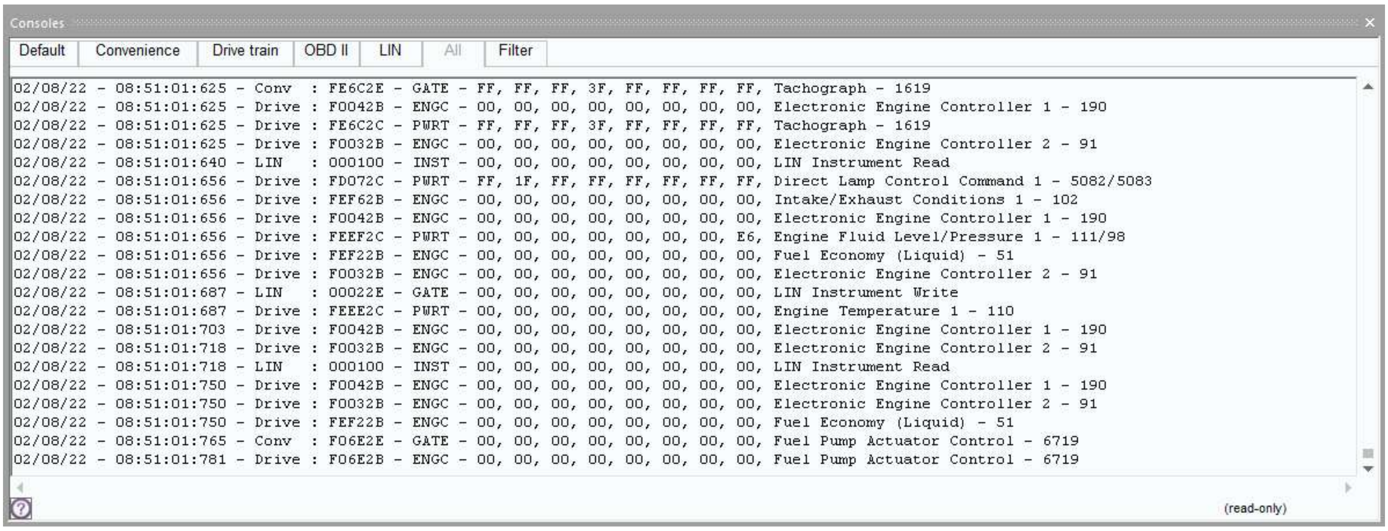
The text fields on the left-hand of the screen show you the collated data on each of the two communications CAN buses, the OBDII bus and the LIN bus. They show the most recent data for each of the PGN messages in the system. There are not many CAN buses in this system so it all fits on the one screen. When sending new messages, this may get untidy - in which case press the Clear button.

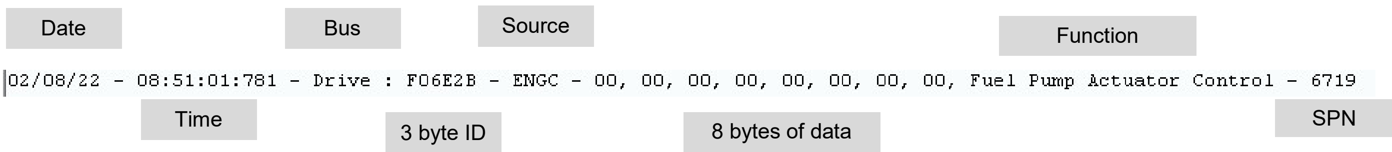
The screen is interactive. When you click on the dials and sensor status fields, the diagnostic software shows you where that information is embedded in the LIN and CAN bus data. This is shown by red boxes that appear on the relevant parts of the message.

This is really useful to help you see the transmission of data in the system and is a little easier to use than the CAN bus look-up table.

Using the console

CAN and LIN bus reference





When you click on the Console button, you get a different view of the message data on the system - the console view.

This shows you the messages on each of the buses. You access each bus by clicking on the relevant tab in the console view.

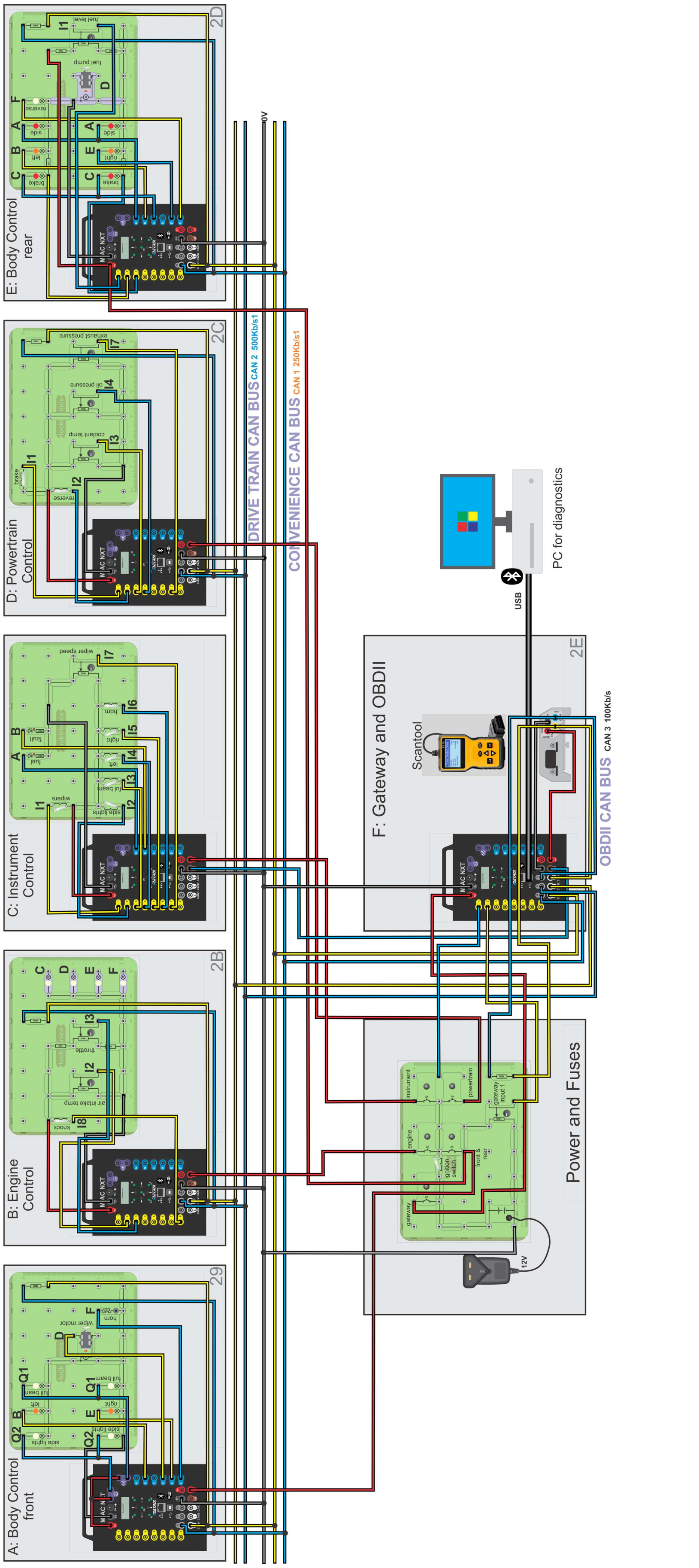
Each message has a time stamp. This is really useful in letting you see in detail the history of the messages on the bus. In particular, it is useful in helping you understand how data is transmitted from one bus to another via the Gateway, for example, from the LIN bus to the Convenience bus.

In addition to the message data the consoles also summarise the different parts of the data as you can see below.

It can be tricky trying to track what happens to the messages. To help, there is a 'Filter' field on the main screen. Click on the filter PGN text and enter the number of the PGN that you are interested in. A filter tab appears showing only those PGN messages on the various buses.

CAN bus system graphic

CAN and LIN bus reference



Version control

First release	01 02 23
Minor changes	26 04 23