

Why You Probably Can Not Make ‘Custom OBD PIDs’ Work for a Recent Car, and What You Can Do About It

Summary

Some manufacturers of recent car models use very effective methods to prevent owners from accessing information about their own cars, even though that information is present in the car network connected to the On Board Diagnostics port. This probably also helps to prevent accidental or malicious damage to the vehicle. Some manufacturers licence the information needed to access the data, to companies that sell OBD tools. This can provide convenient and inexpensive access for owners, while remaining secure if the tool includes features such as Bluetooth pairing only for a short time after pressing a physical button on the tool. This article describes what is possible, using as an example the OBDLink Android app and MX+ interface to the OBD port in a 2019 (gen5) Toyota RAV4 hybrid.

OBD, PIDs?

In recent cars, there will be a port for On Board Diagnostics (OBD-II currently), including ‘legislated’ data items (Parameter IDs or PIDs) that vary according to jurisdiction. The legislated PIDs relate to air pollution, and they will be a subset of those referred to as SAE PIDs, after the standard that first defined them (https://en.wikipedia.org/wiki/OBD-II_PIDs).

The OBD port is usually accessible under the dash near the steering wheel. Data are accessed by plugging in a device called an OBD reader or interface, which acts like a modem between the car and a computer program (which may be a smartphone app). The OBD standards allow for various communication protocols, and evolve over time (<https://obdstation.com/obd2-protocols/>). These protocols may use data from different pins in the OBD port, and some of them allow bidirectional communication. Most interface devices use a command set named after an early manufacturer of chips for the purpose: ELM electronics. This command set has grown with new versions (www.elmelectronics.com/wp-content/uploads/2020/05/ELM327DSL.pdf). OBD standards use 10 ‘Modes’ (hex 01-0A); eg Mode 03 relates to Diagnostic Trouble Codes (DTCs) that are set when a malfunction causes vehicle emissions to exceed legislated thresholds. But many other Modes can exist.

The Challenge in New Cars

Over time, standards have evolved to keep pace with the increasingly sophisticated computer communications and controls in modern vehicles. Most cars now use hundreds or thousands of sensors that communicate through a common set of cables (bus) in a Controller Area Network (CAN) that includes the OBD port. Each sensor sends information into the bus as an identifier (PID) and data bits. The pattern of bits has to be translated into human-readable form through an equation that varies between sensors. Typically there are multiple networks including computer modules called Electronic Control Units (ECUs) that forward necessary information. Some (compound) PIDs give results calculated using inputs from several sensors. OBD standards allow for Headers to address up to eight ECUs, but more exist for other purposes. The wiring for legislated OBD can be used for other purposes that do not interfere. Car makers are not constrained to known ELM commands for corporate uses, which often include vehicle control, and diagnostics through the OBD port. Naturally, some technically-oriented owners would like to access some of this information about their cars!

There are several ‘standard’ CAN protocols (https://en.wikipedia.org/wiki/CAN_bus). CAN11 has $8 \times 16 \times 16 = 2047$ module addresses (Headers) each of which can use multiple Mode and PID combinations for data from sensors. Toyota certainly uses Modes 21, 22 (and maybe others). Mode 21 uses hex $xx = 256$ variant PIDs. Mode 22 allows hex $xxxx = 65536$ variant PIDs. Mode

22 Headers can be long (18DAF130 in the 29 bit example near the end of this article). It is hard to find by chance the Header, Mode and PID for any non-legislated information on the bus.

Manufacturers can implement multiple CAN protocols and/or busses in one car. For non-legislated data they can use proprietary (non-OBD) data formats (eg 11 bit CAN extended addresses, as distinct from 29 bit extended CAN addresses; see *p61 of the ELM command pdf*) and/or keys, passwords, gateway circuits or other means of obfuscation to restrict access (and to prevent accidental or malicious damage to the vehicle).

Non-critical car functions may use cheaper and slower networks, commonly LIN busses. Some of these interface with the CAN network to the OBD port, but some do not. Even if they connect to the OBD port, non-legislated messages may not follow the OBD standard. A module may only respond to requests to its physical address / Header (not the 7DF 'functional' or broadcast Header) or *vice versa*. Even for legislated PIDs, manufacturers are not obliged to allow physical addressing of requests. Message length is always 8 bytes DLC=08 in OBD, but not necessarily in other CAN. Response address is request +8h in OBD, but not necessarily in other CAN. There is so much data on a CAN bus that it is essential to know how to set up appropriate filters for responses. Even if this pattern is deduced, what equation should be used to translate bits for that PID (from sensor voltage) into human-readable results?

Most car manufacturers (OEMs) don't freely release their CAN information. Some go to great lengths to hide PIDs from car owners, particularly in the most recent models. So these PIDs have to be licensed, or discovered by 'reverse engineering' for each model. Depending on the complexity of the bus and the information sought, 'reverse engineering' can take months. People who complete the exercise typically don't freely disclose the results.

The Difficult 'Solution'

There are tools (like an OBD Y-cable and data logger) to record OBD transmissions. Some programs attempt to find (sniff) modules, modes and PIDs available on a connected CAN bus, or [OBD apps](#), but those found may not be supported by a particular OBD app and/or reader. Then the equations must be deduced for translation. All of this is into hacking nerd territory.

The Easy Solution

It is much easier to use OBD software with OEM PIDs. You will need to check that the tool works with the car(s) of interest, provides access to all of the OEM PIDs of interest, and meets your needs for data refresh rate. The cheapest tools generally provide access only to SAE PIDs (if they work at all). But many car makers use 'standard' protocols with OEM-specific PIDs, and some inexpensive apps (eg OBD Fusion) provide access to many OEM PIDs, as do more expensive OEM or licensed 'pro' tools (eg Toyota Techstream, Launch Pro, AutoEnginuity).

OBD Security

Access to your vehicle OBD / CAN network is a huge security threat. For example, a thief with this access can within a few seconds extract the information needed to clone your car 'key' (transponder). With that done, the thief can immediately (or at leisure) enter and drive your car without alarm. Or a person with ill intent could obtain data from the car about your driving, or inject malicious code into any of your car's control computers (ECUs). This is not some far-fetched hypothesis: it has [been done](#).

Wireless (Wi-Fi, BLE or Bluetooth) OBD dongles are an obvious [concern](#), as they can allow any of the above attacks by a person without initial physical access to your car's OBD port. Most cheap dongles allow pairing at all times. Never install a wireless OBD dongle, unless it requires a physical button press to enable pairing. Note that this is just for pairing with an

external device. Pairing takes only a moment, and once paired the external device (like your smart-phone or tablet) can connect with the dongle, and through it your car, any time that dongle is plugged in to your car's OBD port.

Broadcasting to pair at any other time (like every time the dongle is power cycled) is a big drop in security. Do not be deceived by claims that broadcast range or times for pairing are too short to pose a security risk. Bad people who professionally seek access to your vehicle OBD / CAN network have scanners and amplifiers that can provide all they need in a short time at a distance, whenever your OBD dongle broadcasts to pair. You need to control this broadcast with a physical button press at a secure location, only at the one or few times you ever need to pair.

Don't be fooled by [false advertising](#) that OBDLink devices have "Hacker-proof security" that "requires physical access to enable Bluetooth pairing". Firmware after v. 5.6.24 removed this as the default, allowing pairing for a period after each start of the vehicle. Some firmware versions even [prevent setting a safer pairing mode](#) through software. **Double Aarghh!** ☹️🐼🐼

OBDLink: Costs

After trying a cheaper (Panlong) interface with free software (Torque, Car Scanner ELM, OBD Now Terminal) without success for OEM PIDs in a 2019 (gen5, AXAH54) Toyota RAV4 HV, I bought the OBDLink MX+. In June 2021, OBDLink LX and MX+ devices included:

- OBDLink app (= OBD Fusion for Android which costs US\$7). An iOS version is also available.
- OBDWiz program (= OBD TouchScan for Windows US\$30) but only for SAE PIDs without paid add-ons.
- No fee software and firmware updates. The software is being actively improved: check for the latest version.
- Secure (press-button only pairing was advertised) and fast Bluetooth.

MX+ only also included:

- No fee unlimited OEM-specific data add-ons (OEM-Specific Enhanced Diagnostics Support) for the OBDLink app and that specific MX+ unit only (LX or Fusion equivalents = US\$10 each by make and year).

The price in 2019 was US\$80-100 (AU\$170) for the MX+ vs US\$60 (AU\$130) for the LX. These devices have equivalent Bluetooth capability, so unless you access the proprietary Ford (MS-CAN) & GM (SW-CAN) vehicle networks, or use iOs, you would need to use 2-4 OEM-specific downloads to justify the price difference. Costs have since risen substantially (AU\$24 per OEM add-on in 2023).


OBDLink: Warnings

1. On most host devices, these Bluetooth devices are too slow to log multiple OEM PIDs at sub-second intervals. Cheaper USB interfaces may give greater speed, without convenience of wireless.
2. ***OBDLink app after v 5.31.0 may fail to install under Android 7, Aarghh!*** ☹️🐼
3. ***MX+ Firmware after v 5.6.24 vastly reduced security: Double Aarghh!*** ☹️🐼🐼

OBDLink: Features

OBDLink use their own chips (STN****; which also support the ELM327 AT command set <https://www.scantool.net/scantool/downloads/98/stn1100-frpm.pdf>). MX+ tested as ELM v1.4b+ (so it may support some later commands). OBDLink claims advantages of speed (maximum 100 PIDs/sec; though 7-30 OEM PIDs/sec is more like the rate in most host devices), protocol range and firmware upgrade over ELM327. OEM-specific data add-ons provide 'OEM live parameters' for Ford, Nissan, Mazda and Toyota; but only 'OEM DTCs' for others (Holden, Honda, Hyundai etc). Check before purchase: www.obdlink.com/wp-content/uploads/2019/01/app_support.pdf.

Those who want to access data from PIDs in multiple supported cars will find OBDLink MX+ to be a bargain, although OEM PID access is tied to one reader! Alternatively, OEM data purchased in OBD Fusion is tied to one Play Store account. A limitation is that only data from the selected OEM network will show in real time on a dashboard; but in Toyotas most data of

interest is in 'Network A' which loads by default with the SAE PIDs. OBDLink does not have a function that reads all PIDs. You can use Home> Diagnostics> PID Values> Select PIDs to list of those that interest you (in the order they are selected). This is sensible to avoid the many (thousands of) PIDs not of interest at any given time. The easiest way to identify PIDs that may be interesting is to use the  menu to 'select all' PIDs in a module.

I found the OBDLink Android app to be great for gauges (after a few set-up frustrations), though it is too slow for some logging and the Bluetooth connection is lost at times. The Windows program was less impressive, and I did not try iOS. You can not easily use the car 'infotainment' display. A replacement head unit and/or Mirroring Kit is an (expensive) work-around, and it may not work well with all built-in car functions. But you can use an Android pad of a size that fits neatly in the console below the car radio (in the 2019 RAV4 at least, image below). It is safest not to look at either of these places while driving!

OBDLink: (MX+) Set-up (<https://obdsoftware.force.com/s/article/getting-started-with-obd-fusion>)

Using OBDLink (v 5.17) for Android with MX+: (1) Product registration failed repeatedly (cancel out of it); (2) The 'Getting Started' method for OEM add-ons failed, but I eventually bumbled through to fetch them (no other instructions were provided, **see the highlighted tip below**); (3) The automatic firmware update failed (very scary) but after rebooting it was eventually fetched; (4) Sleep (specified to use 1.8 mA instead of 39 mA after 10 min of 'inactivity') worked only sometimes. Pull out the device to ensure no battery drain. (5) The connection process was flaky after several restarts of the car, requiring multiple attempts or restart of the adapter, the app and/or Bluetooth. Pull the MX+ device out of the OBD port when the vehicle is turned off, and re-insert it when needed. (This remained the case under v 5.19, but may be fixed with subsequent MX+ firmware and/or app versions). (6) The app was very slow on each start (or foreground after 'Exit') to reload vehicle data. (7) The Bluetooth connection is sometimes dropped mid-journey. You can work around these frustrations.

I don't know why this device is advertised not to work with hybrid vehicles, but it worked fine with my gen5 RAV4 HV (Australian GX AWD: AXAH54). In contrast, the popular free Hybrid Assistant app did not support this Australian RAV4 model (not OBDLink's fault).

First connection to any Android host device and vehicle:

1. Plug the OBDLink MX+ device into the car OBD port.
Green (power) LED will light and blue (bluetooth) LED will flash.
2. Turn car Ignition On / Engine Off (key or switch position). Use ON or READY in a hybrid.
3. In the phone or tablet, open Android 'Settings', and enable 'Bluetooth'.
4. Press the button on the MX+. The blue LED will blink rapidly, and allow for pairing within 2 minutes. This is really a Pair / Reset button, though some OBDLink docs call it 'Connect'.
5. Open the phone 'Bluetooth settings' menu and tap 'OBDLink device' to pair the device.
6. Tap 'OK' in the phone 'Bluetooth pairing request' dialog box.
7. Start the OBDLink app (OBDLink tells you to get it from the Google Play Store). In Settings> Preferences> Communications, tap 'Bluetooth' (it may be selected automatically).
8. Tap 'Connect' on the OBDLink app home screen.
9. Choose 'OBDLink device'.
10. The app will establish a connection with the OBDLink device (MX+ dongle), and detect which OBD-II protocol your vehicle uses. Once the app establishes a connection with the OBDLink device, the 'BT' LED will display a solid blue light. If you use only one vehicle, it may later be faster to specify the appropriate protocol in Settings, rather than always use 'automatic'.

11. When you connect to any vehicle, OBDLink should allow download of any enhanced OEM add-on (for car-specific PIDs). **No matter where you got the OBDLink app, you have to log in to Google Play Store to obtain data add-ons for OBDLink!** From the OBDLink forum: "Side loading is not supported and getting extras will not work when the app is side loaded without the play store". Rubbish! Side-loading is OK (use an APK repository), and it is necessary for those with Android versions not supported after OBDLink v 5.31.0, but you must still connect to the Play Store to get data add-ons.
12. Then you can select from the various networks in the car, and scan for Supported PIDs (under Vehicle Editor). To eliminate some of the unsupported PIDs, you will need to rescan on each new phone / tablet while connected (eg Distance to Empty disappears under Calculated PIDs, though the RAV4 trip computer has it). When you are not connected to the vehicle, all PIDs are listed.
13. On subsequent use it is only necessary to re-connect the OBD app to the dongle, not re-pair to the host device (android phone or tablet).

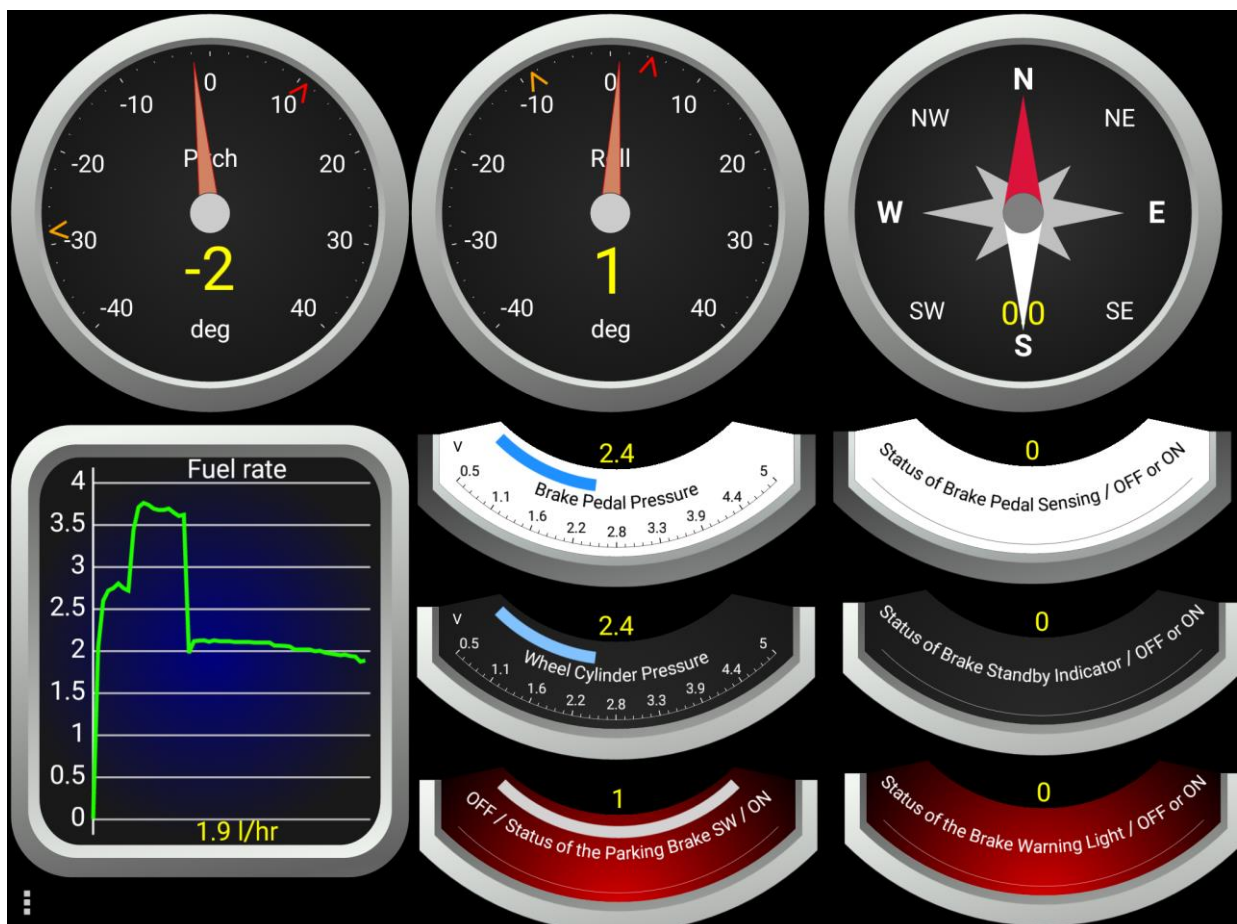
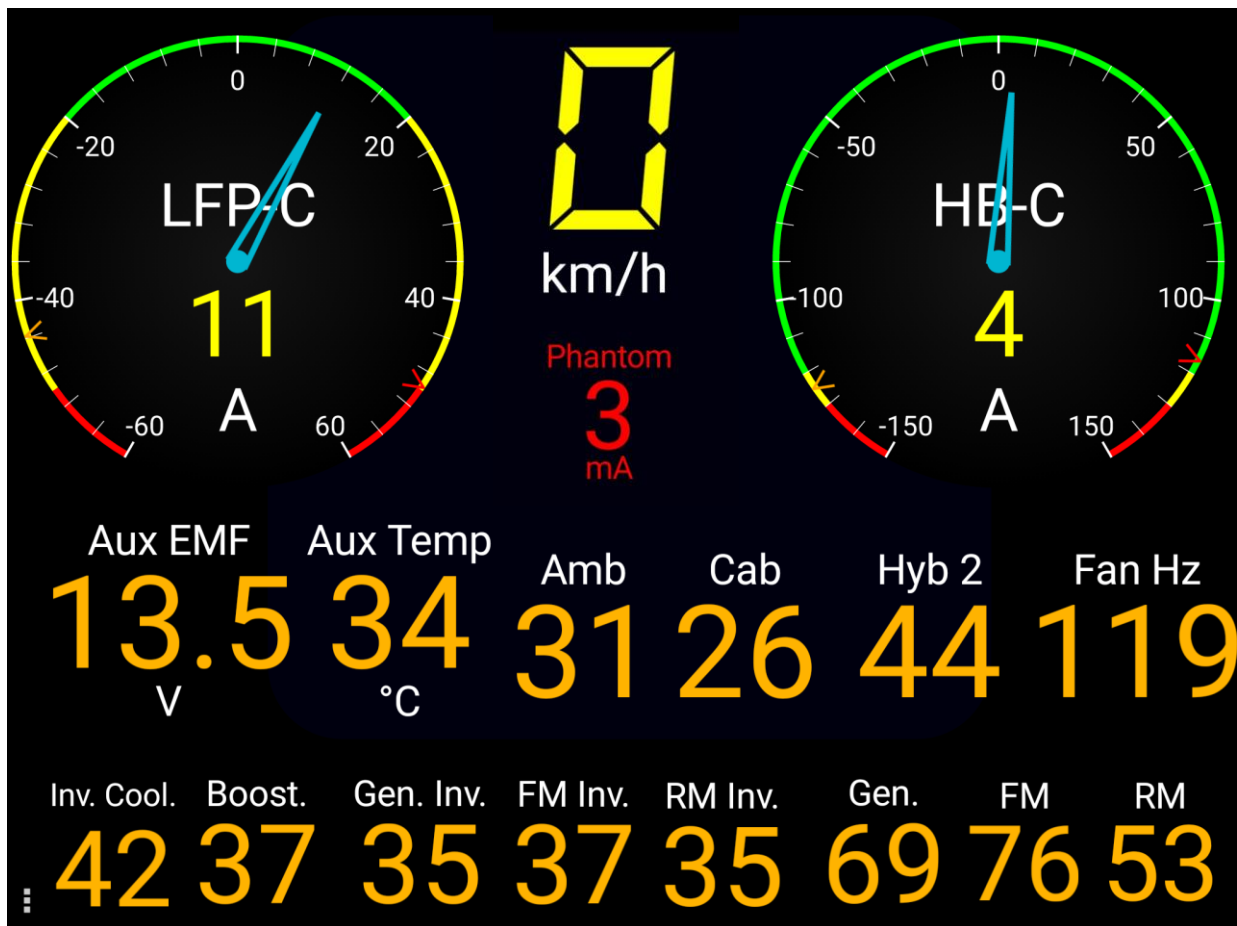
OBDLink: Dashboard Tips (<https://obdsoftware.force.com/s/article/getting-started-with-obd-fusion>)

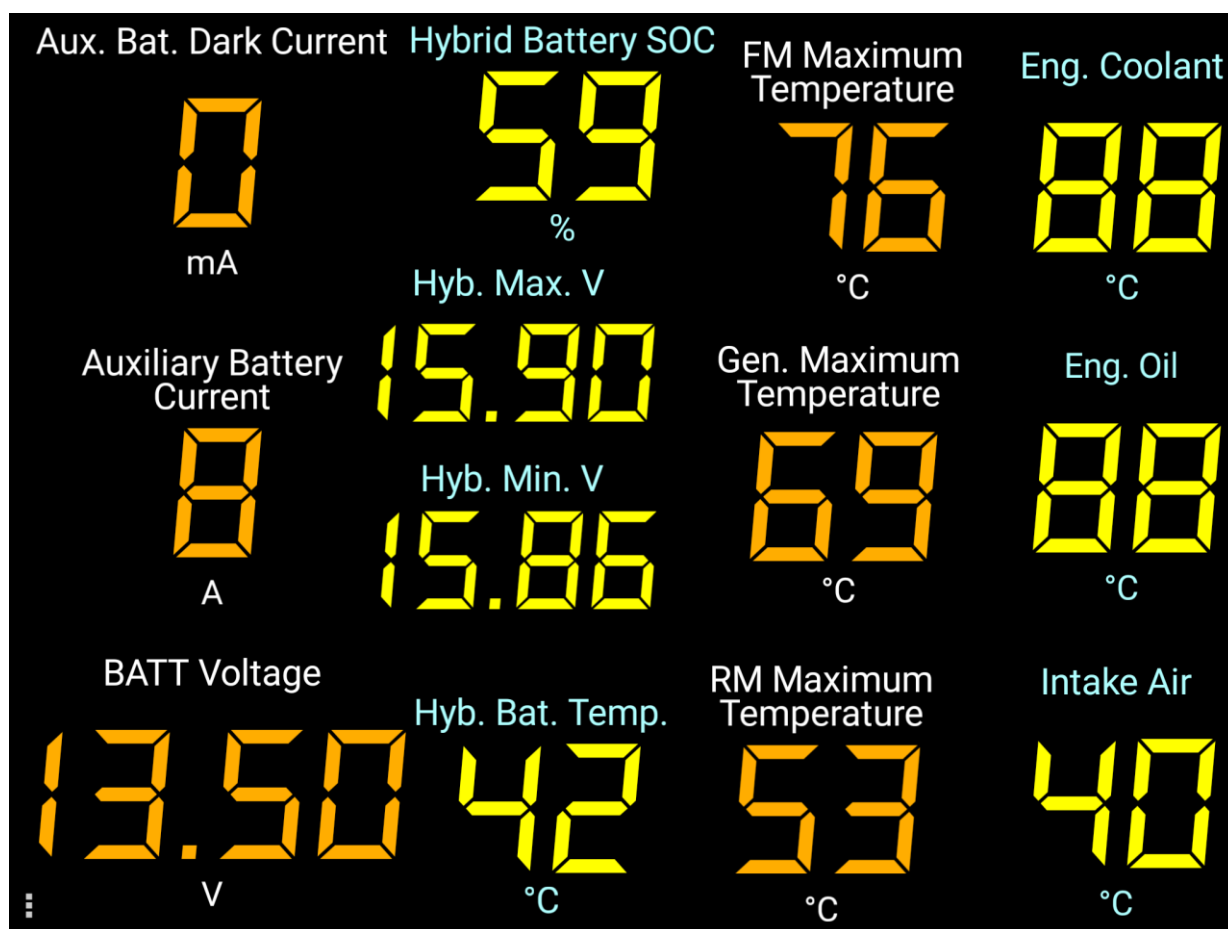
The OBDLink dashboard is very versatile for display of data from any PID in an analogue or digital gauge that makes sense to humans. You can name gauges, create custom bezels, adjust font and size, gauge size, colour, etc. Use 'Range' to set coloured arcs in analogue gauges. From gauge-style> : you can 'save template'. When you start a new gauge, you can select from stock and custom gauge templates. To change an existing gauge you can 'change display type' and/or use gauge-style> : > load template. Home> settings> : 'export settings' includes 'dashboards'. In v 5.19 there is a grid for guidance when you 'drag and move' a gauge. Unfortunately, completed gauges can not be moved or copied between dashboards; and saved templates are not in the exported settings. But any gauge used in an exported dashboard can be saved as template on a new device.

Gauge names auto-wrap (with some bugs) and do not permit added line breaks. Sometimes the desired effect can be achieved with spaces, or with added characters such as ~.

The screen grabs below give an indication of what is possible. Many elegant dashboard designs for OBDLink (Fusion) are posted in car-specific internet forums.







Minor niggles:

1. The word data is plural of datum (OBDLink should say "Data are available", not "Data is available").
2. For navigators, what OBDLink calls **GPS Bearing** is not a Bearing. Rather it is a Heading, Track or Course (terms commonly used as synonyms in navigation of pedestrians and cars):
 - **Bearing** is the direction between any two points (usually the direction from a navigator to a nominated object; such as another vehicle, a landmark or a waypoint). Sometimes relative bearings are used (eg the angle from a ship's long axis), but in navigation the term usually refers to compass bearings. GPS programs may give bearings to a waypoint as magnetic (compass) or true (angle from the meridian connecting N&S poles) - check!
 - **Heading** is the direction the vehicle is pointed. For vehicles like aircraft and ships that may be affected by cross-currents, this can differ from (present) Track or Course, but for cars they are generally the same (except for those who like to 'drift' around corners).
 - **Track** is the actual direction of vehicle travel across the surface of the earth. The term is commonly used in reference to a past track, or a path travelled. It is unambiguous to state a direction in which an object is tracking (as commonly used in aviation).
 - **Course** is the direction in which a vessel is being steered (in shipping), or the intended path of travel to a destination (in aviation). Sometimes these are distinguished as present course vs original course. For a ship in calm weather and no cross-currents, with a steady helm, course=heading.

OBDLink: Toyota AWD gen5 Rav4 HV-specific Tips

Many SAE PIDS (for turbocharged or diesel engines) do not apply. Some that seem to apply (eg Hybrid Charging State) do not function. Manufacturers are not obliged to use all SAE PIDs.

Many of the Toyota PIDs are repeated across modules, with the same sensor (eg engine rpm) used by multiple control algorithms. Some PID readouts are invariant (eg Auxiliary Battery Status of Full Charge). Some do not function (eg Aux Bat Capacity ..., Delta SOC), some are

uninterpretable (eg DC/DC Converter Signal Voltage, Hybrid Bat Check ...), and/or unitless (eg Aux Bat Integrated Thermal Load). Some appear to conflict (eg various Temperatures after the car has been garaged overnight). There are at least four PIDs named Hybrid Bat Current, and they all give (sometimes very) different readings. The HSI PID is labelled %, which it is not, and it does not match the HSI gauge (at left in the Combination Meter below). Many abbrevs are hard to guess. Several sensors may read almost the same thing (eg BATT vs Battery vs Aux Bat Voltage; various Speeds). Without 'inside knowledge' you can only guess at the differences. Some things are strangely named (eg 'Status of the {Front} Motor Temperature' vs 'Rear Motor Temperature') or missing (eg Car GPS & Trip Meter). Some 'normal ranges' are apparently proprietary (eg Inverter Temps). Some designations vary between Toyota documents and the PIDs (eg Generator = MG1, {Front} Motor = MG2, and Rear Motor = MGR).



The in-car displays show regeneration during coasting. Both front and rear motors show slight negative torque, so this may be the signal. There are PIDs for front- and rear- motor regenerative braking (when you use the brake pedal, not coasting). Regenerative brake torque can be controlled (by field-weakening, or by current drawn / circuit resistance). It is evidently specified for a separate location, possibly at the counter-drive gear, where effects from ICE and MG1 also operate. Rear motor negative torque does not change when braking, so this car probably uses slight rear motor regeneration when coasting (little electricity [recovered](#)), but lacks rear motor regenerative braking. On long downhills, the traction battery fills anyhow. Because the brake light is controlled by a switch, not the stroke sensor that controls braking, gentle braking may not operate the stop lights. No wonder the tailgaters get annoyed!

OBDLink Nav vehicle speed averages ~2% faster than SAE speed (probably from wheel sensors): the % varies more at low speeds. Nav speed does not use GPS (it works in tunnels). It may be used in nav functions such as time to destination. Rav4 cruise control and speed alerts use the speed displayed in the MID ('Combination Meter' in Toyota PIDs?). This averages ~3% faster than SAE speed but the readings can be very different. Where the car has a gauge on display, it is generally wise to use the car gauge rather than the reading from an OBD PID.

The gen5 RAV4 NiMH 'traction' battery is specified as nominal 244.8v (220v - 336v while driving) comprising 34 modules (each 6*1.2v=7.2v). There are Toyota PIDs for Hybrid 'Blocks' 1-11, with 3-8 being 4 modules in series and the others 2 modules in series (evident in voltages, which read about 7.7v per module). Hybrid battery voltages should be tested under load (during both charging and discharging while driving) and can be interpreted as in the table (from <https://www.youtube.com/watch?v=k0Mm8XpRVog>).

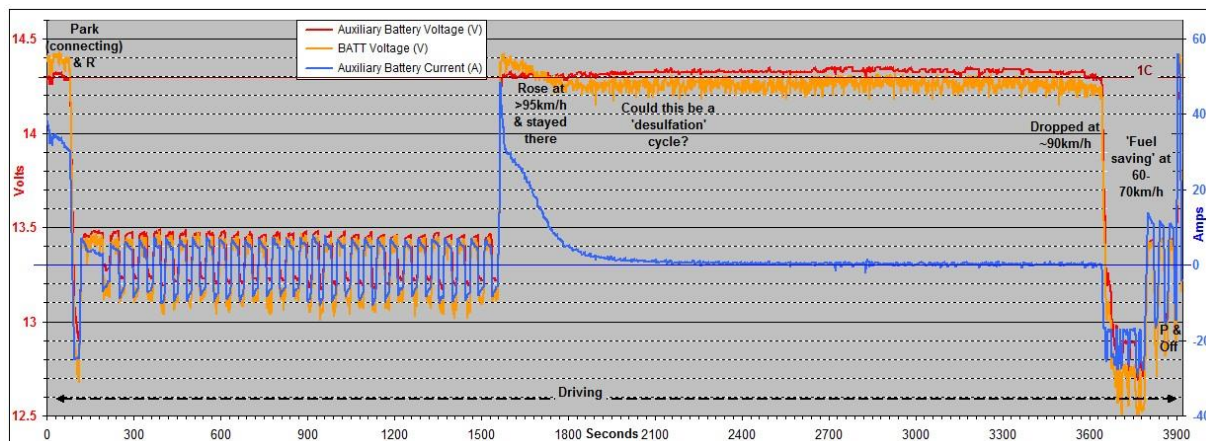
Module? voltage spread (V)	Remaining battery efficiency (%)
0.2	100
0.4	75
0.7	50
0.95	25
1.2	0

The inverter/converter is said (in the [Dismantling Manual](#) for rescuers) to step up to 650v, 3-phase AC for the motors. There are separate motor inverters (in the same PCU).

OBDLINK: Logging Tips (<https://obdsoftware.force.com/s/article/configuring-logging>)

Automatic logging sometimes failed to start on my phone with v 5.17. But it was reliable when I used an Android tablet that was not simultaneously performing other functions (like monitoring a wifi or phone network). Useful new features in v 5.19: (i) a press-button gauge can be used to display / start / stop manual logging; (ii) a start/ stop logging button is on the log files page. Much better! Rarely (after use of another OBD app) it logs only one of the selected PIDs. MX+ does not give anywhere near the advertised 100 PIDs/sec (I get 7 OEM PIDs/sec in Android, or 17 SAE PIDs/sec in Win10, even with dashboard and diagnostic screens off). Even a Panlong (generic OBD interface) is reported as faster on average by the Torque Pro app! Turn off the dashboard and diagnostic screens, and log few PIDs for best [refresh rates](#). Use a USB (not Bluetooth) OBD adapter for faster speeds and fewer inconvenient disconnections.

Logged data can be exported in a csv file. PID columns are in the order they are selected for logging. Excel sees some time values from old versions of the OBDLINK app as text, so it graphs based on categories instead of time values. This problem is reasonably common with other data sources. Unless log intervals are the same as the X-axis units it requires a work-around in Excel (paste special > multiply by a number, typically 1, pre-placed in memory).



OBDLINK: User-defined PIDs (<https://obdsoftware.force.com/s/article/user-defined-pids>)

Headers, Modes and PIDs are not revealed for OEM-specific data add-ons. Therefore, I can not see how to use them in creation of user-defined PIDs (eg combine them in a formula for a calculated PID). There are unexplained settings (like 'Category') for User PIDs in OBDLINK.

It appears that ScanGauge 'reverse engineers' to find some manufacturer-specific PIDs. The 'code' used in early X-gauges is known (<http://www.giffordautomotive.com/images/XGaugeCoding.pdf>). After <http://www.cleanmpg.com/community/index.php?threads/12851/page-2>, the method to 'translate' into PIDs was described more fully in the Torque forum (<https://torque-bhp.com/forums/?wpforumaction=viewtopic&t=352.0>), then widely copied (and sometimes elaborated or mistaken). However, X-gauge coding has 'evolved'. For example, what does C mean at the start of RXF? The most recent X-gauges work only with new ScanGauge firmware, and the changed methods are not freely disclosed.

I have tried but never succeeded in setting up a non-SAE User PID for a 2019 RAV4, even 'translating' from X-gauges. But unless you want to create some kind of compound PID, why bother with 'User PIDs' when 'OEM PIDs' are available? Car manufacturers do not generally change PIDs every year in the same series or generation of each car model. So if PIDs for the most recent year are not yet available, it is worth trying to manually set an earlier year of the same model, and download the OEM data for that year in OBDLINK.

Use of an OBDLink Device with a Terminal Emulator

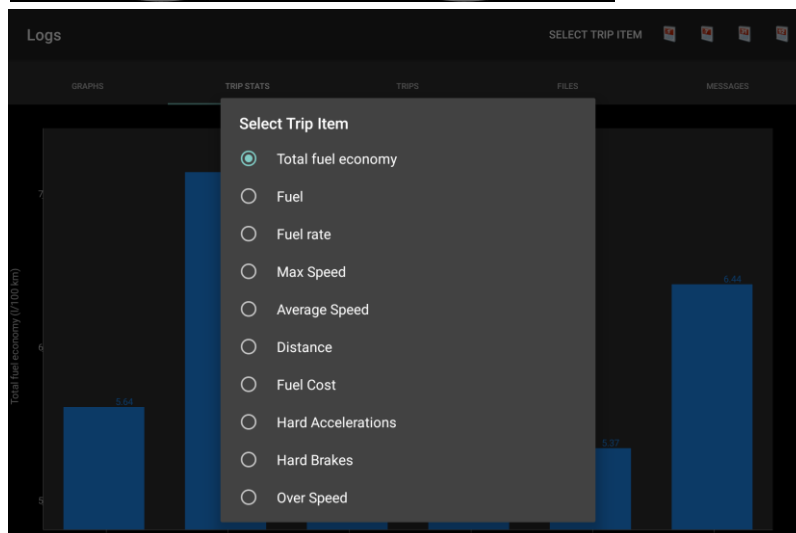
The OBDLink devices can be used to communicate with the car via a terminal emulator app (<https://www.glmsoftware.com/documentation/OBDNowTerminalUserGuide.pdf>), or the same function built into apps like OBDwiz, but unless you know the Toyota PID #IDs (and maybe other methods used to obfuscate the process) it is much easier to just use the OEM add-on in the OBDLink app.

For requests of data from legislated SAE PIDs via a terminal emulator, the gen 5 RAV4 HV seems only to allow functional addressing (request Header 7DF but not 7E0, 7E2 or 7E6), even though results come from modules with response Headers 7E8, 7EA and 7EE. Cheaper OBD interface devices may also work for this more limited purpose.

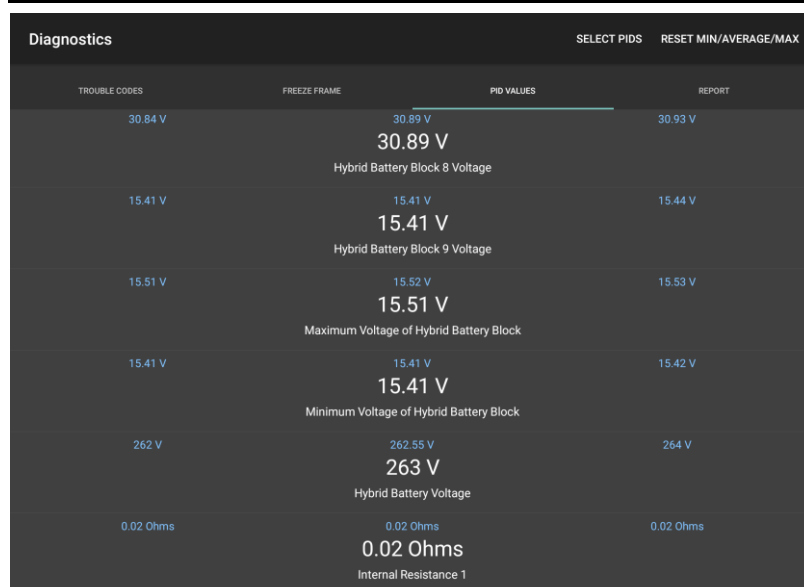
OBDLink: Useful Undocumented Features



New versions sometimes bring new features, but **versions after 5.31.0 may not work with some Android devices**. With some devices (eg a Samsung Tab S2) OBDLink can report Pitch and Roll (under Sensors), and allows these to be set to zero for any device location (under the dashboard menu: Calibrate Device Sensors).



Home >
Logs >
Trip Stats
has lots of interesting
information.



Home >
Diagnostics >
PID Values
can give interesting real-time
data from all selected PIDs
(with min/ av/ max during the
current connection or reset
period).

OBDLink: Auto-Sleep/Wake, Connect and Log

<https://www.scantool.net/forum/index.php?topic=16242.0>

Auto-wake and Auto-reconnect do not work reliably with some older versions of MX+ Firmware and OBDLink Android App; even with a dedicated Samsung Galaxy Tab S2 (no wifi or phone but Bluetooth on), 'Automatic Connection' selected, 'Prompt for ECU' not ticked, 'Start with Last Connected Protocol' ticked, +/- 'Connect Silently', +/- 'Dashboard Page' in a 2109 Gen5 Toyota RAV4 hybrid (Australian GX AWD).

In ST mode the MX+ sleep time is 10 min of UART inactivity. Toyota PID "Auxiliary battery dark current" will reveal ~40mA drain from MX+ in idle, or ~2mA in sleep. OBD device LEDs:

- Power (green): **Solid – Ready**
1 second blink - Booting; Brief blink every 3 seconds - Sleep
- OBD / Host: Blink for any OBD / Host communication
- BT (blue): Solid - Connection is active
Blink - Waiting for connection; Fast blink - Waiting for pairing

According to OCTech (developers of OBDLink and Fusion / Touchscan apps): "If a supported Bluetooth LE adapter is detected within a brief duration after launching the app, a connection to the scan tool will be automatically initiated." whereas: "The app will store the SSID of the Wi-Fi network. If the phone is connected to this network when launching the app or when the app is running in the foreground, connection will be automatically initiated."

When the OBDLink app is first launched or restarted after 'exit', it takes a long time to load vehicle data from an OEM add-on; then it does auto-connect to a paired OBDLink interface device (like MX+) if I turn on both: (a) Settings>Preferences>Communications>Connection Mode>Automatic, and (b) Settings>Preferences>General>Dashboard>Start on Dashboard Page.

But when the OBDLink app is then brought to foreground on vehicle restart (after MX+ sleep), it usually can not reconnect to the MX+ and go to the dashboard page without manual intervention (screen clicks). There are several ways to re-connect, so YMMV. In any case, the app must be taken off the dashboard screen or sent to background if the android host device is to sleep. All of this defeats users seeking the advertised "Fully automatic" operation.

The free (& add-free) [Automate](#) app will do the job: if the host device is not set to lock, and the ECU prompt is turned off in OBDLink settings and Vehicle Editor. The "[flow](#)" I use currently performs the following tasks:- When the car is turned on: Enable BT > Foreground OBDLink app > Delay 1s > Click 'connect' > Delay 8s awake > Click 'Dashboard' > Monitor power source. When the car is turned off: Android 'back' > Delay 1s > Click 'disconnect' > Delay 2s > Disable BT > Background OBDLink app (allow host device to sleep) > Monitor power source. This works, with MX+ sleep, in with OBDLink app v 5.31.0 and MX+ firmware v 5.7.0.

This makes the re-connection truly automatic (no manual input required) provided that MX+ is plugged in before the car is started. It has the added advantage of turning BT off when not needed, for longer battery life (and greater [security](#)) in the display device. If the "flow" does not work as expected; eg if OBDLink decides to reload OEM data, or if a delay is too short, or if MX+ is not plugged in, or if the car is turned on in 'ignition' instead of 'ready' mode; you will need to intervene manually to restart the "flow". Preferences in the OBDLink app should be set up before using this "flow" for automation. Automate (v 1.35) can not detect Bluetooth SPP devices (like MX+) in 'waiting for connection' mode, or directly connect them through apps like OBDLink. So the "flow" relies on simulated screen clicks that are a method of last resort (because this method does not work on some android host devices, and such devices vary in speed and screen dimensions). Automate uses very little power while running in the background.

It might be simpler if the OBDLink app developers could make the automatic re-connection and dashboard features work on foregrounding (when the device is powered by starting the car), then disconnect from the OBDLink interface device and background the app so the host device can sleep when the car is turned off. It is probably a difficult job to make the OBDLink app suit both those who use an OBD-dedicated android device (wanting automatic re-connect), and those who use a multi-purpose device like a phone (wanting no interference with other functions of their device). Perhaps the OBDLink app developers can specify launch options, broadcasts or some-such that can be used in Automate to have the OBDLink app: (i) change to a Dashboard Page; (ii) connect to a paired OBDLink device; (iii) disconnect, or otherwise stop searching before a timeout when the vehicle is turned off and (iv) return to the background.

After connection, logging usually works well with 'Automatic' selected under 'Logging Control'. Rarely it will log only one of the selected items. This did happen after I had tested the Dr Prius app on another phone, so I can not be sure if that left any temporary residual effect.

Auto-connection on starting the car is very handy, and this makes sense for those who use an OBD-dedicated android host device (an old phone or tablet) in the vehicle, paired to the interface device. For this currently I use the Automate App "flow" mentioned above, with OBDLink App settings: (a) Preferences>Communications>Connection Mode>Manual, and (b) Preferences>General>Dashboard>Do not Start on Dashboard Page. It may also be necessary to load [extensions](#) and use 'native' in the 'timing accuracy' settings in Automate. It is not hard to find the RAV4 OBD port by feel, and unplugging MX+ there ensures zero battery drain, so no chance of long-term car battery damage. But the MX+ (with firmware 5.7.0) goes to pairing mode for a few minutes after each power on, which is a security risk.

OBDLink: Toyota OEM PID Names

Of the Toyota PIDs, OBDLink (in its forum) says 'Toyota provides the list' and 'the app data used to pull the PIDs is provided directly by Toyota'. The PIDs have names (which can be quite cryptic) but no descriptions or #IDs. So I peeked into Techstream, which seems intended to provide detailed PID descriptions under Data List Manager > Parameter Information.

Usually all I see there (in TS versions 13, 14, 16 & 17) is a copy of the PID name.

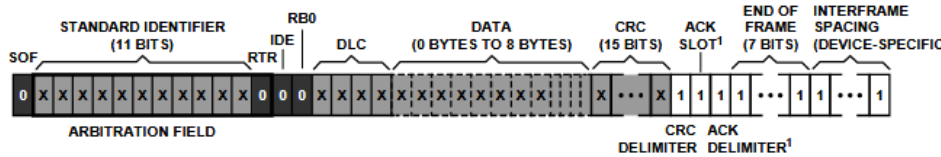
Some PIDs display an extra word or two (check some ABS PIDs), but nothing helpful.

Parameter	Value	Unit
Auxiliary Battery Voltage	12.96	V
Auxiliary Battery Voltage Low Times	0	
Auxiliary Battery Voltage at Low Voltage Checking Initiation	12.57	V
Auxiliary Battery Charging Integrated Current	1649.5	Ah
Auxiliary Battery Discharging Integrated Current	702.4	Ah
Auxiliary Battery Capacity after IG ON	0	Ah
Auxiliary Battery Capacity after IG OFF	0	Ah
Integrated Ready ON Time	117	hour
Number of Long Term Leaving with IG OFF	0	
Auxiliary Battery Integrated Thermal Load	6179.2	km
Total Distance Indicated after Long Term Leaving with IG OFF (1st)	0	km
Total Distance Indicated after Long Term Leaving with IG OFF (2nd)	0	km
Total Distance Indicated after Long Term Leaving with IG OFF (3rd)	0	km
Time of Long Term Leaving with IG OFF (1st)	0	day
Time of Long Term Leaving with IG OFF (2nd)	0	day
Time of Long Term Leaving with IG OFF (3rd)	0	day
Auxiliary Battery Average Current during IG OFF 1 Trip before	-0.217	A
Auxiliary Battery Average Current during IG OFF 2 Trip before	-0.305	A
Auxiliary Battery Average Current during IG OFF 3 Trip before	1.998	A
Auxiliary Battery Average Current during IG OFF 4 Trip before	-0.046	A
Auxiliary Battery Average Current during IG OFF 5 Trip before	-0.101	A
Total Distance Up to 1 Trip before	25468	km
Total Distance Up to 2 Trip before	25432	km
Total Distance Up to 3 Trip before	25426	km
Total Distance Up to 4 Trip before	25388	km
Total Distance Up to 5 Trip before	25388	km
IG OFF Time before 1 trip	0	day
IG OFF Time before 2 trip	0	day
IG OFF Time before 3 trip	0	day
IG OFF Time before 4 trip	0	day
IG OFF Time before 5 trip	0	day
IG ON Time Up to 1 trip before	64	min
IG ON Time Up to 2 trip before	10	min

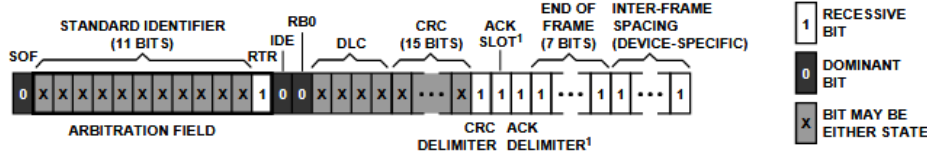
Parameter Information
 Auxiliary Battery Integrated Thermal Load
 Total Distance Indicated after Long Term Leaving with IG OFF (1st)
 Total Distance Indicated after Long Term Leaving with IG OFF (2nd)
 Total Distance Indicated after Long Term Leaving with IG OFF (3rd)

Example of CAN data

STANDARD DATA FRAME



STANDARD REMOTE FRAME

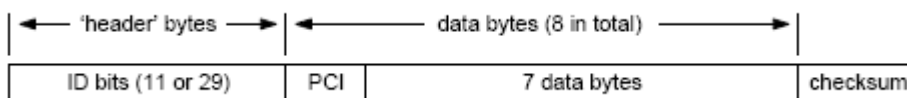


KEY:
 1 RECESSIVE BIT
 0 DOMINANT BIT
 X BIT MAY BE EITHER STATE

NOTES

1. ORIGINATOR OF FRAME TRANSMITS RECESSIVE (1) DURING ACK SLOT/DELIMITER. SUCCESSFUL TRANSMISSION OF MESSAGE FRAME REQUIRES AT LEAST ONE OTHER NODE TO TRANSMIT A DOMINANT (0) BIT DURING THE ACK SLOT.

CAN Standard Message Frame Fields



A CAN OBD Message

<https://fabiobaltieri.com/2013/07/23/hacking-into-a-vehicle-can-bus-toyothack-and-socketcan/>

Modified from comment by David (2015/06/26). Using as an example:

18DAF130 X | 8 | 02 7E 00 55 55 55 55 | 5076

- The (29 bit) message ID (in hex) is 18DAF130. 18DA is a standard identifier for physically addressed signals (asking one module for a response). An 11 bit message ID would use the physical ID of the module (7E* for SAE). You may also see 18DB (7DF in 11 bit messages) which is functional addressing. In [29-bit](#), F1 lets you know a "tester" is receiving this information. 30 is the sender's address. This could be any module on that particular bus (but it is probably the [steering ECU](#), see SAE 32178/1). The X marks an extended message (29 bit identifier).
- 8 (DLC) is the length of the message data to follow (in bytes: standard in CAN is 8 bytes).
- This is the data that you actually care about (in hex): 02 7E 00 55 55 55 55
- 02 is the length of the data in this message that is significant (not counting itself)
- 7E 00 is a [UDS-on-CAN](#) identifier indicating a "tester present acknowledged" message
- The 55s are just padding (the 02 in the first [PCI byte](#) showed two more bytes of significant data).
- In short, this message was a response from a module on the bus to the tester, in response to a tester present message (not included in your data dump) using a higher-level UDS protocol. Multiple modules (30, 28, 10, 53 and 60) responded (maybe to a broadcast 02 3E 00 [heartbeat](#))

This is the interesting message (18DB33F1 would ask all modules for data):

18DA30F1 X | 8 | 03 22 48 00 00 00 00 | 417576

- This is the tester asking module 30 for some Mode\$22 data from PID 4800. There are a bunch of different modes (or services) that do [different](#) things; some proprietary to the car maker.
- Module 30 then responded (3B is hex for 59 **bytes in response**; 10, 21 & 22 are PCI bytes of multiline response indicators; 30 in a PCI byte is a flow control indicator, with F1 telling 30 about any needs for timing; the response always adds 40 hex to the request mode, so 22→62):

18DAF130 X | 8 | 10 3B 62 48 00 00 00 F8 | 4435
 18DA30F1 X | 8 | 30 00 00 00 00 00 00 | 484
 18DAF130 X | 8 | 21 C0 00 00 00 00 52 14 | 1483
 18DAF130 X | 8 | 22 11 21 01 00 00 00 08 | 1466

...

The information contained within this multi-frame response is a secret of the car-maker.

Message Arbitration on a 'Classical' CAN bus

CAN bus systems (used for OBD in all cars since 2008) resolve conflicts (simultaneous message transmission) based on message ID/priority. A low bit has priority. The CAN standard does not require node addresses and may use message IDs based on anticipated urgency, for more efficient use of bus. The OBD standard uses node addresses as the basis for message ID (header). As a node transmits one message at a time, this meets the need for arbitration.

In OBD on 11 bit CAN (where the tester/OBD reader is assumed to be at hex address F1):

- the initial 0 of a hex address converted into bits is omitted, to fit within 11 bits
- broadcast (functional) requests from a tester have 7DF as ID
- requests to one ECU ('physical requests') have the address of the target ECU as ID
 - but not all ECUs accept physically addressed requests, even for SAE PIDs
- responses have the 'physical' address of the target ECU + 08h as ID

IN OBD on 29 bit CAN:

- there is space for a 5 bit 'priority byte', always set to hex 18 for diagnostics
- broadcast (functional) requests have ID 18DB33F1
- requests to one ECU have 18DAxxF1 as ID (where xx is the address of the target ECU)
- responses have 18DAF1xx as ID (where xx is the address of the responding ECU)

These methods may change under other CAN protocols ('CAN extended', CAN FD, CAN XL proprietary variants, etc).

