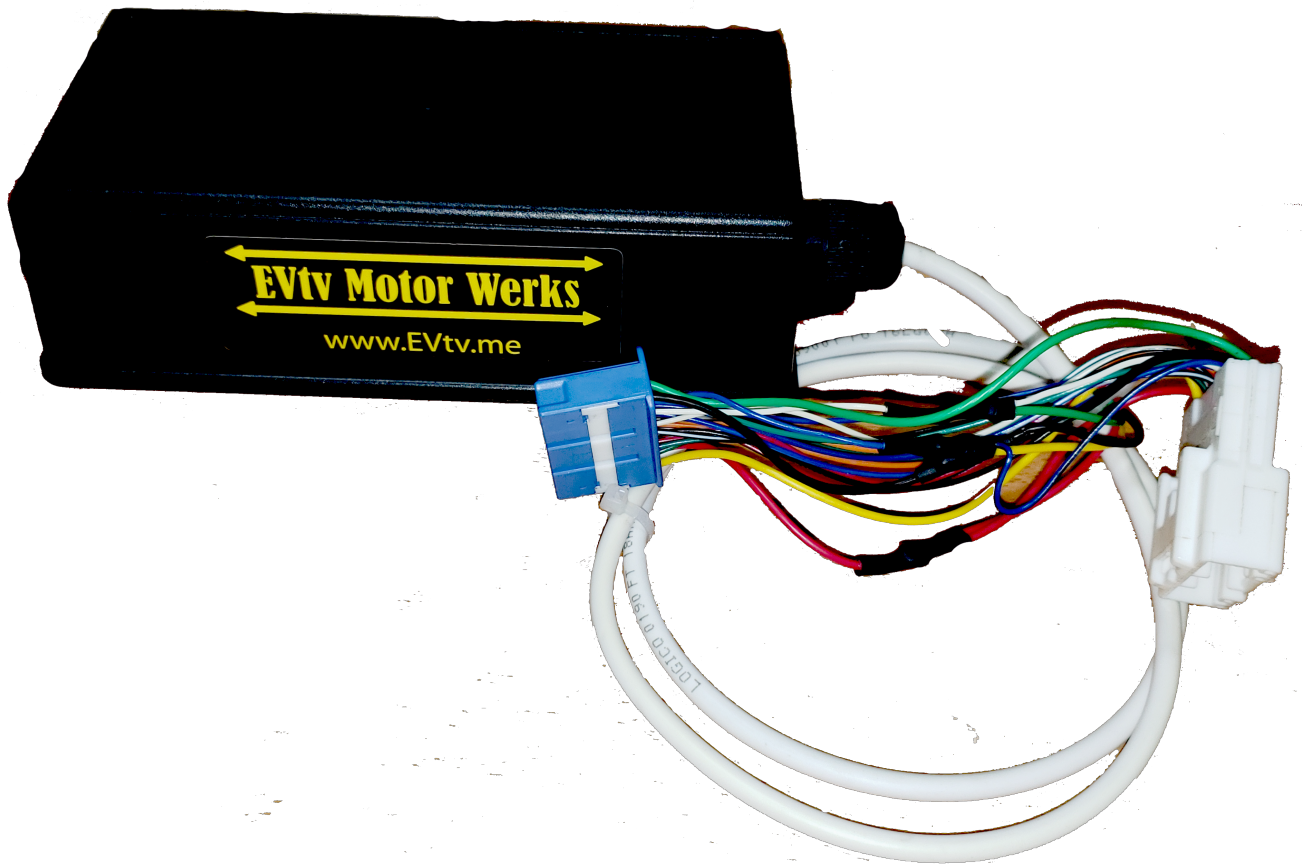


User Manual

CAN Adapter for Tesla Model 3



INTRODUCTION

The EVTV Tesla Model 3 CAN Adapter allows you to easily capture and analyze Controller Area Network (CAN) Traffic from the Tesla Model 3 Vehicle CAN bus.

Analyzing the Tesla Model 3 CAN bus is challenging in several ways. First, unlike Tesla's Model S and X, the Model 3 provides neither an OBDII connector nor a CAN diagnostics port. We must insert a harness between two existing connectors to intercept signals from the vehicle.

Second, these 500kbps CAN busses are heavily loaded – we've logged frame rates as high as 2000 frames per second on a 500 kbps bus. This is far beyond the abilities of most of the inexpensive OBDII CAN tools available and even beyond the reach of some commercial devices without dropping frames.

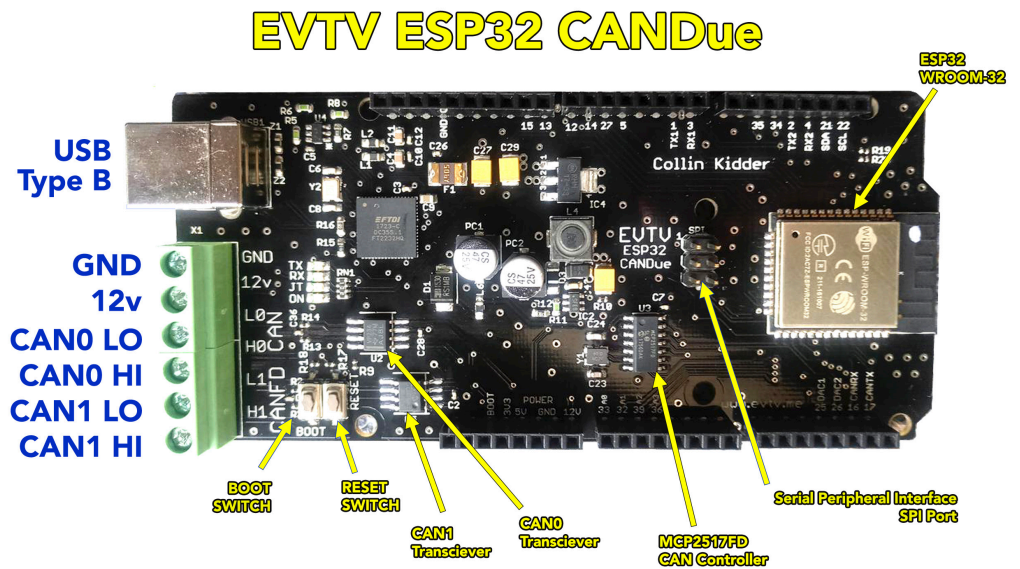
The EVTV Tesla Model 3 CAN Adapter includes:

1. EVTV ESP32 CANDue Microcontroller with a Tesla Model 3 harness/connector.
2. ESP32 Reverse Engineering Tool (ESP32RET) software to run on the microcontroller.
3. SavvyCAN CAN Data Analysis software to run on any laptop or desktop computer – Microsoft Windoze, Linux, or MAC OSX.

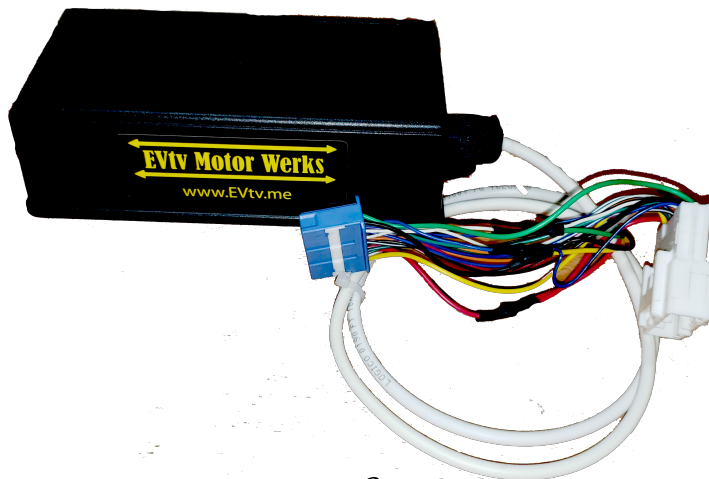
EVTV ESP32 CANDue Microcontroller

The EVTV ESP32CANDue Microcontroller hardware consists of a 240MHz Dual Core Espressif ESP32 microcontroller that can be easily programmed in C++ using the free and open source Arduino IDE.

The board features two CAN bus transceivers. It is housed in a plastic enclosure with a Mini B Universal Serial Bus printer port for connection to a laptop and an external harness to mate with the Tesla Model 3 x930 connector in the vehicle console.



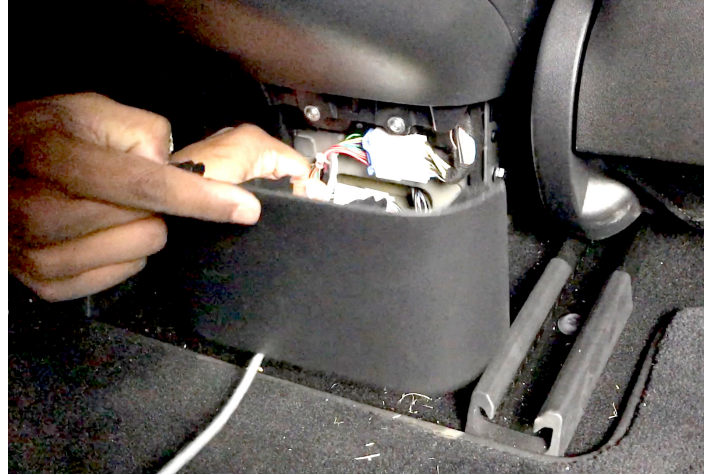
The Espressif ESP32 chip features an onboard radio to connect wirelessly with a variety of devices using both Wireless 802.11x for TCP/IP and UDP connections and Bluetooth BLE commonly used to communicate with smart phones and tablets.



INSTALLATION

To install the EVTV ESP32CAN Model 3 CAN Adapter:

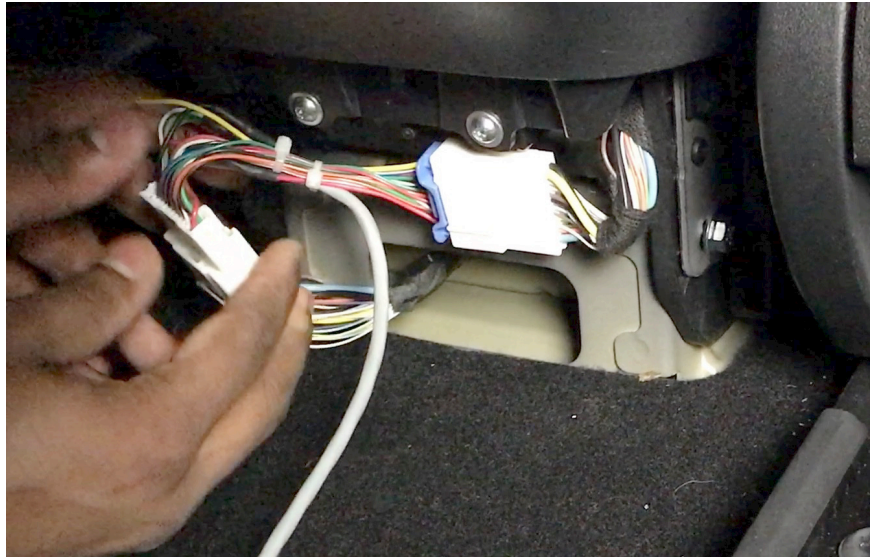
1. Remove the plastic cover assembly from the lower portion of the rear face of the console. You may need to pry gently around the edges with a flat screwdriver to gain an edge to pull on. Pull it firmly to the rear.



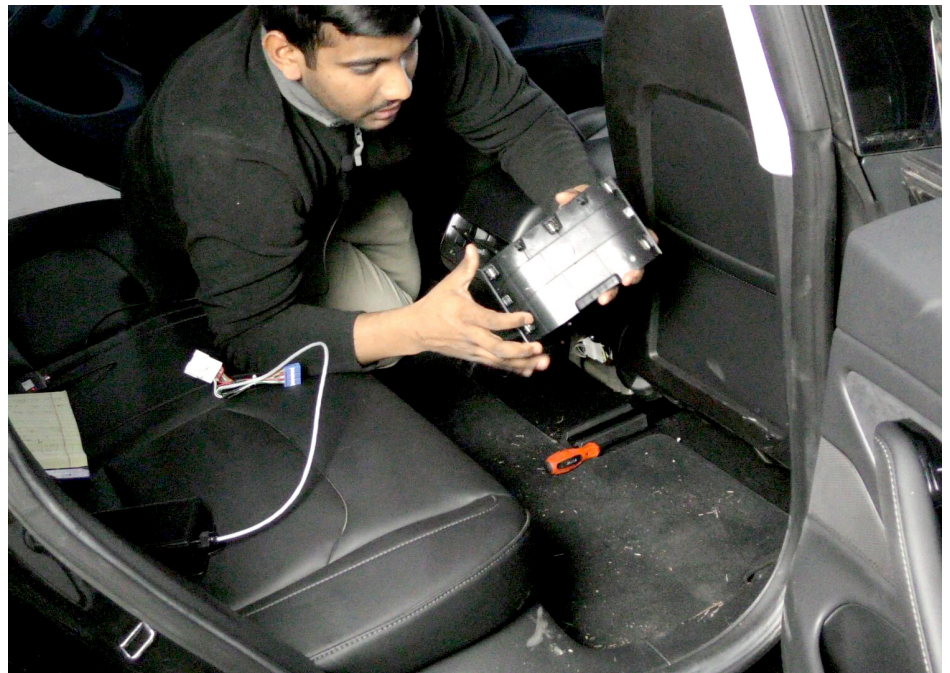
2. Disconnect connector 0x930 by pressing the small white tab and separating the connectors.



3. Insert the harness assembly connectors into each of the two 0x930 connectors.



4. Carefully place the cover into position aligning the five seating pins and push firmly into place.



SOFTWARE

The Tesla Model 3 CAN adapter comes with the ESP32 Reverse Engineering Tool (ESPRET) software installed. You can connect any laptop with an ordinary serial terminal screen to the USB port of the device to connect directly to the device if you like.

Entering **?** on the command line and pressing ENTER will cause a menu screen to come up. You can use this to manually select CAN bus selections, speeds, etc.

Normally, this will not be used by most users of this product. Updates can be obtained from <https://github.com/collin80/ESP32RET>.

A much more capable CAN analysis program is available for using the adapter from a laptop computer via USB port. It is titled SavvyCAN and features a number of advanced CAN analysis functions. But it makes it very easy to log off CAN traffic and save it to files, load them later for analysis, and examine them with a number of views.

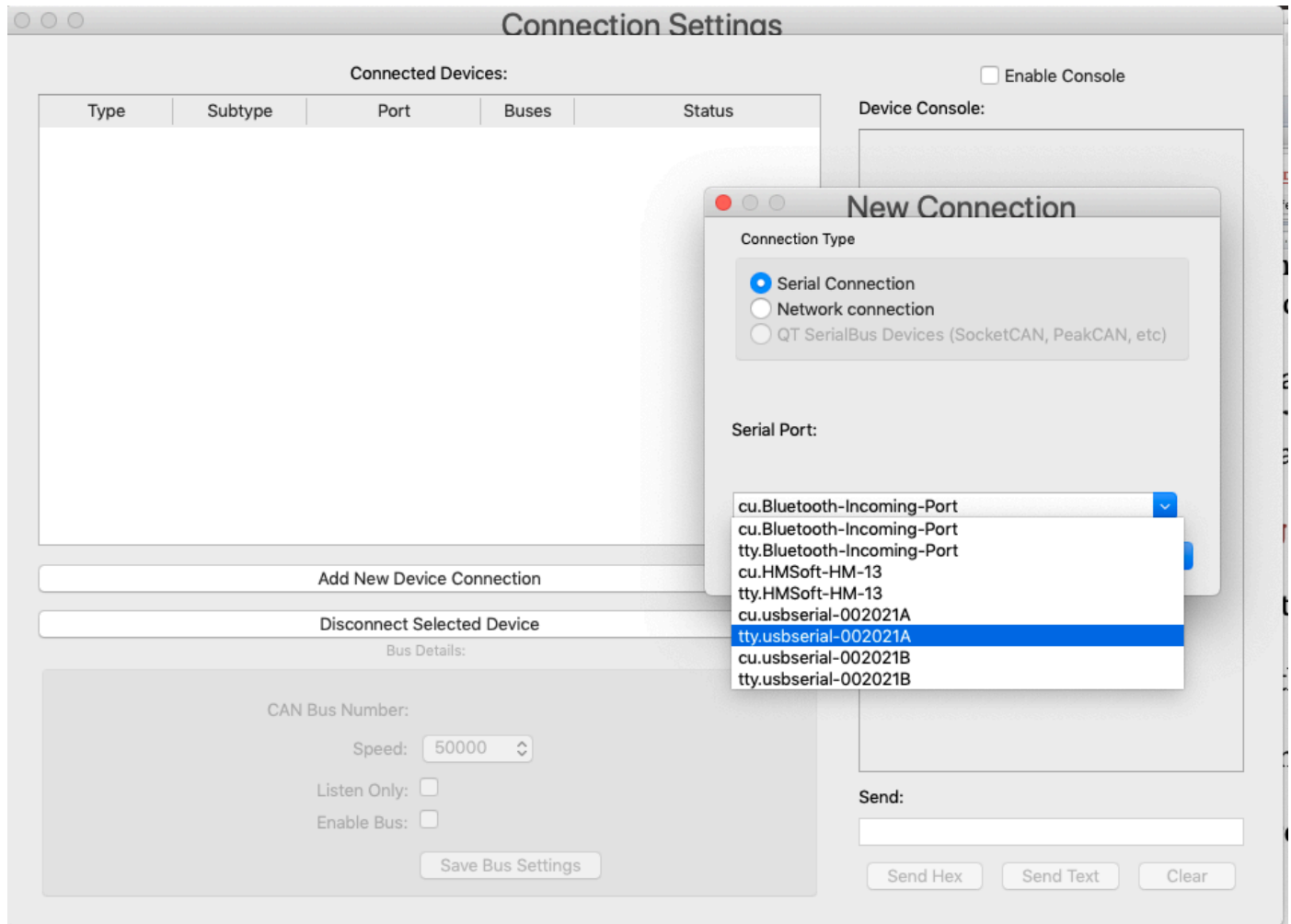
SavvyCAN is available for Windows, Mac OSX, and Linux. It exchanges messages with the Tesla CAN adapter to send and receive CAN message traffic automatically. <http://www.savvycan.com>

Links for both SavvyCAN and GEVRET are available at the EVTV store website under the detailed view of the Tesla CAN bus adapter product. These programs are under continuous development so for best results ensure you have the latest versions installed.

Either program can also be found at <http://github.com/collin80>.

The basic process of capturing CAN data is pretty straightforward:

1. Connect your laptop to the USB port on the EVTV Model 3 CAN adapter
2. Start the SavvyCAN software program on your laptop.
3. In the upper right hand corner of the screen, select CONNECTIONS to bring up the connections window.



4. Select Serial Connection and the serial port used to connect to the adapter.
5. Select CAN BUS 0 and 500000 as the Speed. Then ENABLE for the bus.

NOTE

SavvyCAN and ESPRET use the very generic USB port configurations common on Linux or Mac OSX right out of the box. However, some Windows installation may require installation of USB serial port “drivers” in order to function properly.

The easiest way to do this is download and install the Arduino IDE. This automatically configures your Windows machine for the necessary USB serial port configuration.

Savvy CAN V141

| Timestamp | ID | Ext | Bus | Len | Data | |
|-----------|----------|--------|-----|-----|------|--|
| 28 | 0x022814 | 0x0228 | 0 | 0 | 4 | 0x40 0xC0 0x30 0xFB |
| 29 | 0.022976 | 0x0125 | 0 | 0 | 4 | 0x02 0x00 0x31 0x09 |
| 30 | 0.023226 | 0x03AC | 0 | 0 | 8 | 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 |
| 31 | 0.023806 | 0x0154 | 0 | 0 | 8 | 0x10 0x07 0xC2 0x1F 0x40 0x00 0x00 0x00 INVMSG4 |
| 32 | 0.026095 | 0x032C | 0 | 0 | 8 | 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 |
| 33 | 0.026923 | 0x021C | 0 | 0 | 8 | 0x00 0x08 0x00 0x00 0x85 0x01 0x00 0x14 |
| 34 | 0.027157 | 0x025C | 0 | 0 | 8 | 0x01 0x02 0x26 0x9A 0x4F 0x00 0x3A 0x07 |
| 35 | 0.027678 | 0x0102 | 0 | 0 | 6 | 0x58 0x97 0x35 0x80 0x73 0x00 ACCEL_DATA LATERAL: -0.297194G |
| 36 | 0.028928 | 0x0E | 0 | 0 | 8 | 0x1F 0xEA 0x3F 0xFF 0x08 0xFF 0x30 0x43 STEER_POS STEERINGPOS: 304.2% |
| 37 | 0.029161 | 0x0106 | 0 | 0 | 8 | 0xA8 0x9F 0xA8 0x1F 0x05 0x09 0x19 0x3C INVMSG1 MOTOR_RPM: -7901 MOTOR_TORQ_FILTERED: 255.4ft-lbs MOTOR_TORQ_RAW: 49.6ft-lbs THROTTLE: 47.04% |
| 38 | 0.029689 | 0x0116 | 0 | 0 | 6 | 0x00 0x40 0x75 0x43 0x84 0x93 INVMSG2 |
| 39 | 0.030286 | 0x0368 | 0 | 0 | 8 | 0x00 0x5A 0x1F 0x00 0x00 0x08 0x00 0xFF |
| 40 | 0.030544 | 0x0708 | 0 | 0 | 8 | 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0xFF |
| 41 | 0.032714 | 0x0228 | 0 | 0 | 4 | 0x40 0xC0 0x40 0xA1 |
| 42 | 0.032959 | 0x03BC | 0 | 0 | 8 | 0x00 0x55 0x55 0x55 0x01 0x00 0x00 0x00 |
| 43 | 0.034146 | 0x0154 | 0 | 0 | 8 | 0x10 0x07 0xC2 0x1F 0x40 0x00 0x00 0x00 INVMSG4 |
| 44 | 0.035722 | 0x031E | 0 | 0 | 6 | 0x00 0x00 0x00 0x00 0x00 0x00 |
| 45 | 0.035838 | 0x040E | 0 | 0 | 1 | 0x00 |
| 46 | 0.036329 | 0x033C | 0 | 0 | 8 | 0x00 0x55 0x55 0x55 0x01 0x00 0x00 0x00 |
| 47 | 0.03724 | 0x0102 | 0 | 0 | 6 | 0x4B 0x97 0x3E 0x80 0x3C 0x00 ACCEL_DATA LATERAL: 0.298759G |
| 48 | 0.03899 | 0x0106 | 0 | 0 | 8 | 0xA8 0xBF 0xA8 0x1F 0x07 0x09 0x19 0x5E INVMSG1 MOTOR_RPM: -7901 |
| 49 | 0.039227 | 0x0E | 0 | 0 | 8 | 0x1F 0xEA 0x3F 0xFF 0x08 0xFF 0x40 0x1A STEER_POS STEERINGPOS: 304.2% |
| 50 | 0.039425 | 0x0116 | 0 | 0 | 6 | 0x00 0x40 0x76 0x43 0x85 0x95 INVMSG2 |
| 51 | 0.040038 | 0x0718 | 0 | 0 | 8 | 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 |

Serial Port
HMSoft-HM-13

Connect To GVRET

CANBus Speeds:
First Bus: Disabled Second Bus: Disabled

Set CANBUS Speeds

Total Frames Captured:
311882

Frames Per Second:
0

Suspend Capturing

Normalize Frame Timing

Clear Frames

Auto Scroll Window
 Interpret Frames
 Overwrite Mode

Frame Filtering:

- 0x0E
- 0x0102
- 0x0106
- 0x0108
- 0x0116
- 0x0125
- 0x0126
- 0x0138
- 0x0154
- 0x0168
- 0x01F8
- 0x0202
- 0x020A
- 0x020C
- 0x020E
- 0x0210
- 0x0212
- 0x0218

All None

Failed to connect! bridgeacceleration.csv loaded teslamodels_bus3_v5.dbc loaded.

At this point the SavvyCAN display should start showing CAN traffic in the DATA field. Make sure your Model 3 is “started” and the dash instrument displays are live to ensure flow of CAN traffic. You are now “recording” CAN traffic from the Tesla Model S drive train to memory.

TOTAL FRAMES CAPTURED

This displays the total number of message frames received from the adapter and held in memory.

FRAMES PER SECOND

This is the rate of frames being received per second.

SUSPEND CAPTURING

Pressing this button simply discontinues capturing frames from the port to the memory. It also changes the button to **RESUME CAPTURING**. In this way, you can simply stop and start frame captures at will.

NORMALIZE FRAME TIMING

Normally, the frames are displayed with the time stamp applied by the GVRET device on receipt. This is the number of microseconds since the device was first powered up. But you might want something more in line with the capture. Click NORMALIZE FRAME TIMING to set the FIRST frame received as time 000. All subsequent frame times will be in relation to that frame.

CLEAR FRAMES

This button simply resets the logging memory to zero frames – effectively erasing everything you’ve captured so far. You can use CLEAR FRAMES to clear memory and then RESUME CAPTURING to start a fresh capture at any time.

AUTO SCROLL WINDOW

Normally, the data window simply displays the first 25 frames or so and you can “scroll” down with the mouse to view subsequent frames. This checkbox allows you to autoscroll the window to the left so that the newest data always appears as the bottom line of the screen. In this way you are always viewing the LAST 25 frames received. You can still scroll up and down.

INTERPRET FRAMES

Interpret Frames allows you to examine values contained in CAN frames. It performs any necessary math functions on the data for you to get the real value. For example, Voltage might be in bytes 1 and 2 of a frame in LSB/MSB format, and multiplied by 100. Interpret Frames would then take byte 2 *256, add the value in frame 1, and divide the total by 100 to get voltage.

These functions are defined for each frame in a Vector Graphics format .DBC file. Think of this as a rules file or library for all the different data definitions you have. This file can be LOADED by using

the LOAD DBC file on the top bar FILE menu. They can also be SAVED using the SAVE DBC option on the same menu.

DBC files are normally created and edited separately.

OVERWRITE MODE

Overwrite mode shows all the UNIQUE message IDs and overwrites each with the latest data as it is received. Works in capture but not on a loaded data log.

FRAME FILTERING

Frame filtering is a very powerful function allowing you to define specifically which frames are displayed. The window beneath lists all unique frames received so far. The ALL button at the bottom puts a check box next to each message ID assuring that they will be captured. The NONE button removes all checkboxes.

You can of course put check marks next to any message IDs of interest and those will subsequently be displayed.

Note that all frames are still captured to memory. The filter simply determines which frames are displayed.

FRAME NUMBER

Leftmost column on the display lists the number of the message frame received. Each incoming frame is assigned a subsequent number.

TIMESTAMP

The time at which the associated frame is received starting with the beginning of capture. As noted, this can also be “normalized” with the first frame at time 00000.

ID

The 11-bit or 29-bit message identification number from the received frame.

EXT

This will display a 1 if 29-bit extended frame was received or 0 if standard frame

BUS

Some adapters can support two CAN busses and SavvyCAN can receive CAN traffic from two busses simultaneously. This column shows which bus (usually 0 and 1) that the associated message was received on. This will always be 0 for the Tesla Model S capture device.

LEN

Number of data payload bytes in this message.

DATA

This shows the actual data payload bytes received in hexadecimal format. If INTERPRET FRAMES is on it may also show additional data showing what is contained in those bytes as interpreted by the DBC file rules.

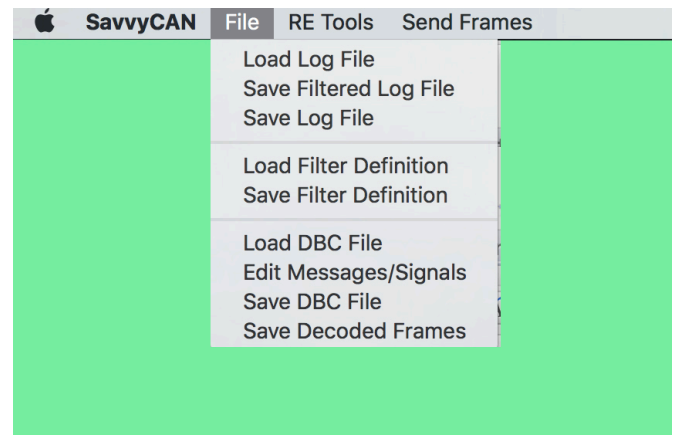
FILE MANAGEMENT

In addition to the main communications screen, SavvyCAN has a top bar menu allowing access to other functions and indeed other analysis screens.

SAVE LOG FILE

The FILE menu lists a number of options allowing you to save and load data in various files.

SAVE LOG FILE is the most important function of SavvyCAN. It allows you to save all the CAN traffic you have captured to your hard drive in a variety of file formats for later use by SavvyCAN, or by other programs or spreadsheets.



LOAD LOG FILE

Load logfile of course is the other end of this. In addition to CAPTURING CAN data, SavvyCAN provides many analysis tools that can be applied to previously captured log files as well. LOAD LOG FILE simply allows you to reload a previous capture into memory for analysis.

SAVE FILTERED LOG FILE

As previously described, the filter window on the main capture screen simply determines what messages are DISPLAYED – all messages being retained in memory. But you may truly only be interested in a few messages. SAVE FILTERED LOG FILE allows you to save a data log of JUST the frames of interest. In this way, the next time you load that file, it will contain only the messages of interest and will of course be a much smaller file as well.

SAVE AND LOAD FILTER DEFINITION

Similarly, there may be quite a bit of work developing your message ID filter where you have hundreds of possible messages, and several dozen that you ARE in fact interested in. This function allows you to save a filter definition, and later reload it without having to manually check each box again.

LOAD/SAVE DBC FILES

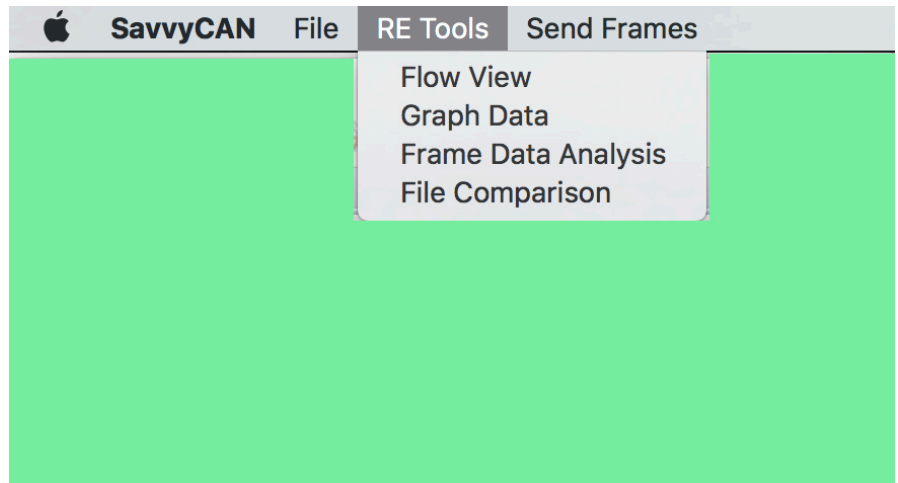
DBC files are simply a file format developed by Vector Graphics to save data definitions of CAN data in a library file allowing you to interpret those frames later using the definitions. The file format simply ends in **.dbc**.

SavvyCAN actually supports the Vector data definition files and you can import and export .DBC files from other programs into SavvyCAN. You can also EDIT MESSAGES/SIGNALS and SAVE DECODED FRAMES – that is messages for which the data definition is known.

REVERSE ENGINEERING TOOLS

SavvyCAN goes quite beyond capturing and saving CAN messages from the bus. It provides a variety of data analysis tools that can be of huge assistance in reverse engineering what messages do what on a CAN bus.

These tools are available in the RETOOLS menu of the SavvyCAN top bar menu.



FLOW VIEW

Flow view is provided to allow you to select a single message ID and then follow its flow through an entire CAN capture session, noting the changes from frame to frame for that single message ID.

You basically “play” back the capture recording while focusing on a single frame.

Up to 8 data bytes are displayed and you can easily compare their numeric contents to the startup state or to the previous frame.

Flow View also graphs each of up to 8 data bytes in the payload.

And finally, a 64 bit array of all 8 data bytes and each bit of each byte is graphically represented on screen so you can see individual bits wink in and out as it changes from frame to frame.

FRAME IDs FOUND

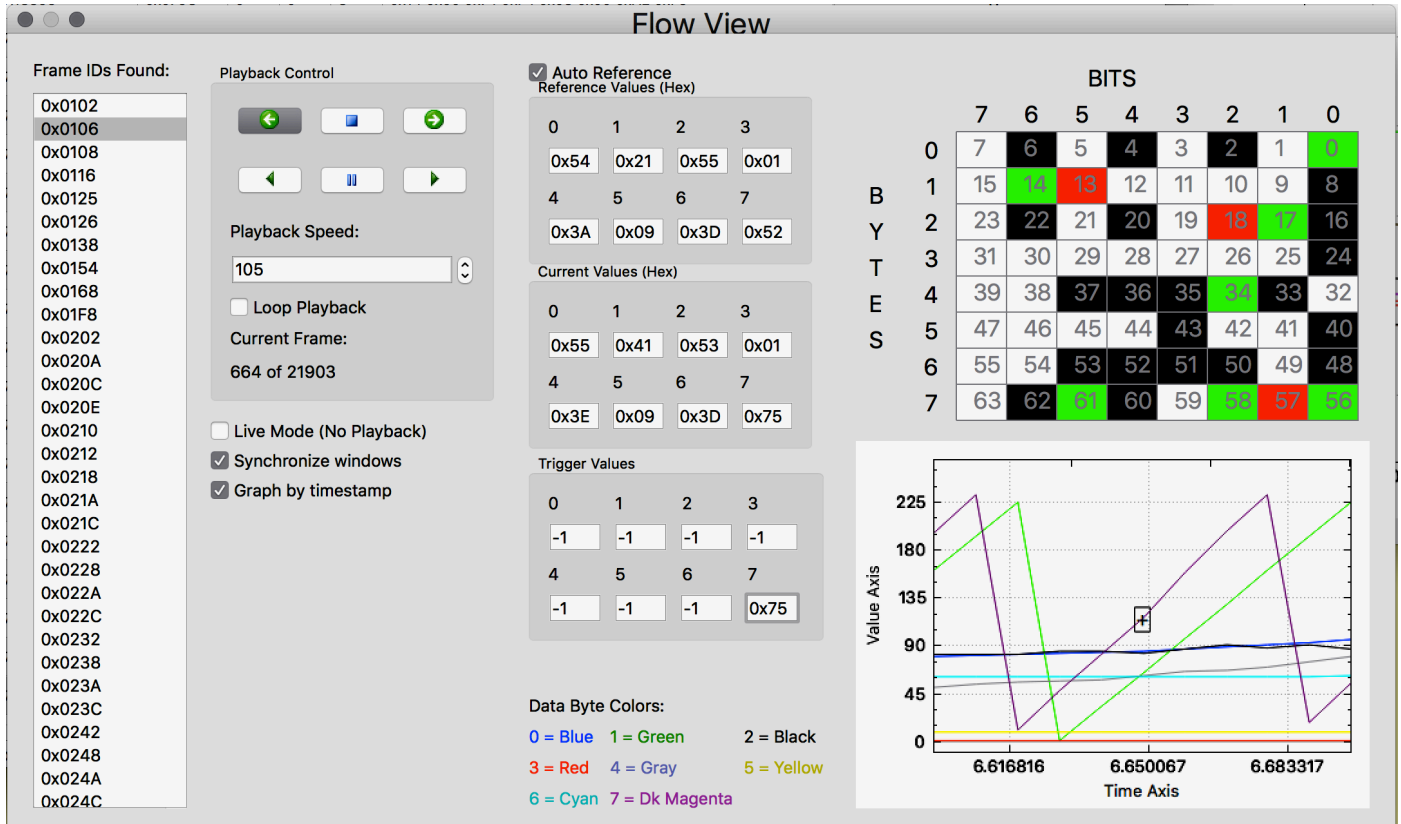
This box lists all unique CAN messages available in the current capture file in memory. You select the frame to examine with Flow View by highlighting one of these message IDs.

PLAYBACK CONTROL

The familiar playback controls allowing you to play through the series of CAN messages of that ID, either automatically or manually stepping through them one at a time.

PLAYBACK SPEED

Allows you to vary the rate at which automatic playback occurs.



LOOP PLAYBACK

With this checkbox, once end of file is reached, you can simply start over with the first message captured with that message ID.

CURRENT FRAME

Shows you total frames of this message ID in the file and the frame number of the current frame displayed from this file.

SYNCHRONIZE WINDOWS

When you “play through” the sequence of occurrences of a byte in flow view, if this checkbox is elected, the play in flow view is synchronized and linked with the action in the main SavvyCAN serial communications panel, AND with the play through in the GRAPH VIEW screen. So that at any particular moment, all three views are displaying the same value. This lets you STOP the action at any particular point in the graph view of the entire capture, and actually see the values of the message ID under scrutiny in flow view.

GRAPH BY TIMESTAMP

Normally, the frames for any particular message ID are numbered for this view and indeed on the left we display the current frame number and total number of message frames of that ID. The small graph in the lower right hand side of the panel normally graphs the 7 data bytes by frame number. By clicking this, the graph will show TIME along the horizontal axis instead, making it easier to compare our location with the GRAPH function elsewhere or the main data view. Shows graph of each data byte by time stamp of time message received.

AUTOREFERENCE

Allows you to either list the initial frame from the file or the previous frame displayed as the reference frame which you are comparing current frame to.

The central display area of the screen shows the reference frame and the current frame with hexadecimal values for each current byte. These are updated as the frames are played.

DATA SEEK VALUES

Data seek values allows you to set seek points in the series of messages to stop the playback. The playback will run until that specific value is found in that byte. At that point it will stop. You can then change speeds or make other adjustments and then continue.

DATA BYTE COLORS

Lists the colors used to graph each data byte in the graph display to the right. This running graph updates as the frames are viewed in their "flow".

BITS

Bits displays each data byte from 0 to 7 in the data payload. It further breaks out each bit of those bytes as bit 0-7. This forms a 64-bit display grid.

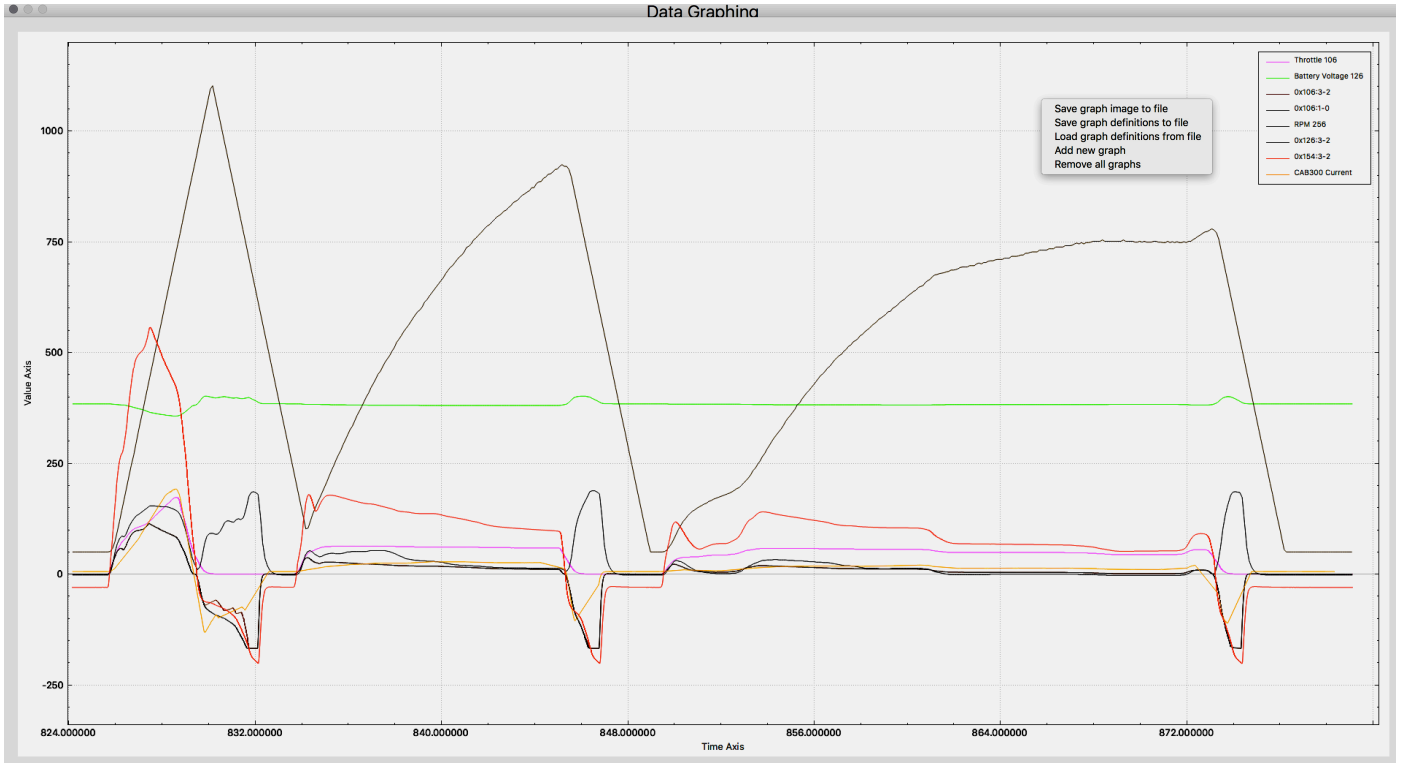
Bits set in the FIRST frame of this message ID are displayed in black.

Red denotes bits that WERE black,, but are currently reset (0).

Green denotes bits that WERE NOT set initially, but now ARE.

GRAPH DATA

Data graphing is a powerful analysis tool allowing you to graph data values over time. Better, you can graph several of them over the same time frame. This lets you examine relationships between different data as they interact. For example, as you see your torque increase, indicating a large



Save graph image to file
Save graph definitions to file
Load graph definitions from file
Save spreadsheet of data
Add new graph

Reset View
Zoom In
Zoom Out

demand for current, you may notice a smaller dip in another value that is concurrent. That value may turn out to be voltage. But only by seeing the two interact can you make that determination.

To create a graph, press CNTRL and click on the graph screen to bring up the graph menu.

This screen allows you to do a number of things. You can load and save a graph definitions file containing your graph definitions.

You can also save a graph IMAGE file of the graphic image as it appears on screen.

You can save a spreadsheet of the data used on the graph.

But most importantly it allows you to add a new graph.

This is a new data value which will be graphed on the form on screen. And you can add any number of these to the screen.

When you select ADD NEW GRAPH, you will see a new control box titled GRAPH SETTINGS. This is where you enter values defining what and how to graph data.

NAME is simply the name of the value shown in the legend.

ID is the message ID of the CAN message you wish to graph.

DATA is the crucial element. It defines the byte or bytes to be graphed. This is a value between 0 and 7 indicating a byte containing a value of interest.

You can enter multibyte values as in 2-3 where bytes two and three are treated as a 16-bit integer. In the case of LSB/MSB values, you can enter 3-2 to indicate that.

SIGNED indicates if you want to treat this as a signed integer or unsigned integer.

MASK allows you to mask off individual bits of the value to ignore.

BIAS allows you to offset the graph vertically from the zero origin line.

SCALE allows you to multiply or divide the value by any number – effectively scaling the size of the waveform.

STRIDE allows you to graph every nth data point instead of each one. For example, enter 10 to graph every 10th message frame of that ID. For some large data sets or messages that occur very frequently, this can declutter the display and improve performance on large data sets.

And **COLOR** will allow you to define what color the graph line will appear.

You can click on the timeline below to expand or contract the graph in time. And you can click on the values to the left of the graph to expand or contract the range of values depicted.

You can also click/drag on the central graph area to zoom BOTH time and value in and out.

In this way, you can add multiple data elements from the same or different CAN messages all to the same graph. You can zoom in and out and examine their relationships in value and time.

The screenshot shows a 'Graph Settings' dialog box with the following fields and controls:

- Name: CAB300 Current
- ID: 0x03C2
- Data: 0-3
- Signed:
- Mask: 0xFFFFFFFF
- Bias: -2.14748e+06
- Scale: 0.001
- Stride: 1
- Color: CHANGE (button)
- Message: (dropdown menu)
- Signal: (dropdown menu)
- Add this graph (button)

And you can save your definitions or even an image of the graph to a file. You can load the definitions later to graph the same data from a different log file.

FRAME DATA ANALYSIS

Select Frame Data Analysis from the RE TOOLS menu to call up the Detailed Frame Information screen.

This screen provides statistical data on each specific CAN message received.

All received message IDs are listed on the left under the Frame IDs and indeed this panel displays the total number of unique message IDs received in this capture or data log.

Highlight any message ID to display statistical details in the right hand panel.

In this example, we highlight message ID 116 and see that the capture log contains 21,903 instances of this message, that the data length in the 116 message is 6 bytes, and that the average time interval between receipts of this message is 9999 microseconds or about one message ever 10 milliseconds.

If we expand Data Byte 1 we learn that the value in byte 1 ranges from 0x10 to 0xC0. And a histogram provides the number of times each value appears in the data log.

The most common value is 0x40 which appears in 20,064 messages.

A Bitfield Histogram actually shows the number of times each bit of the data payload is true. We see no bits in Byte 0 are ever lit, but bit 22 (byte 2, bit 6) is set 12,049 times in the 21,903 messages received.

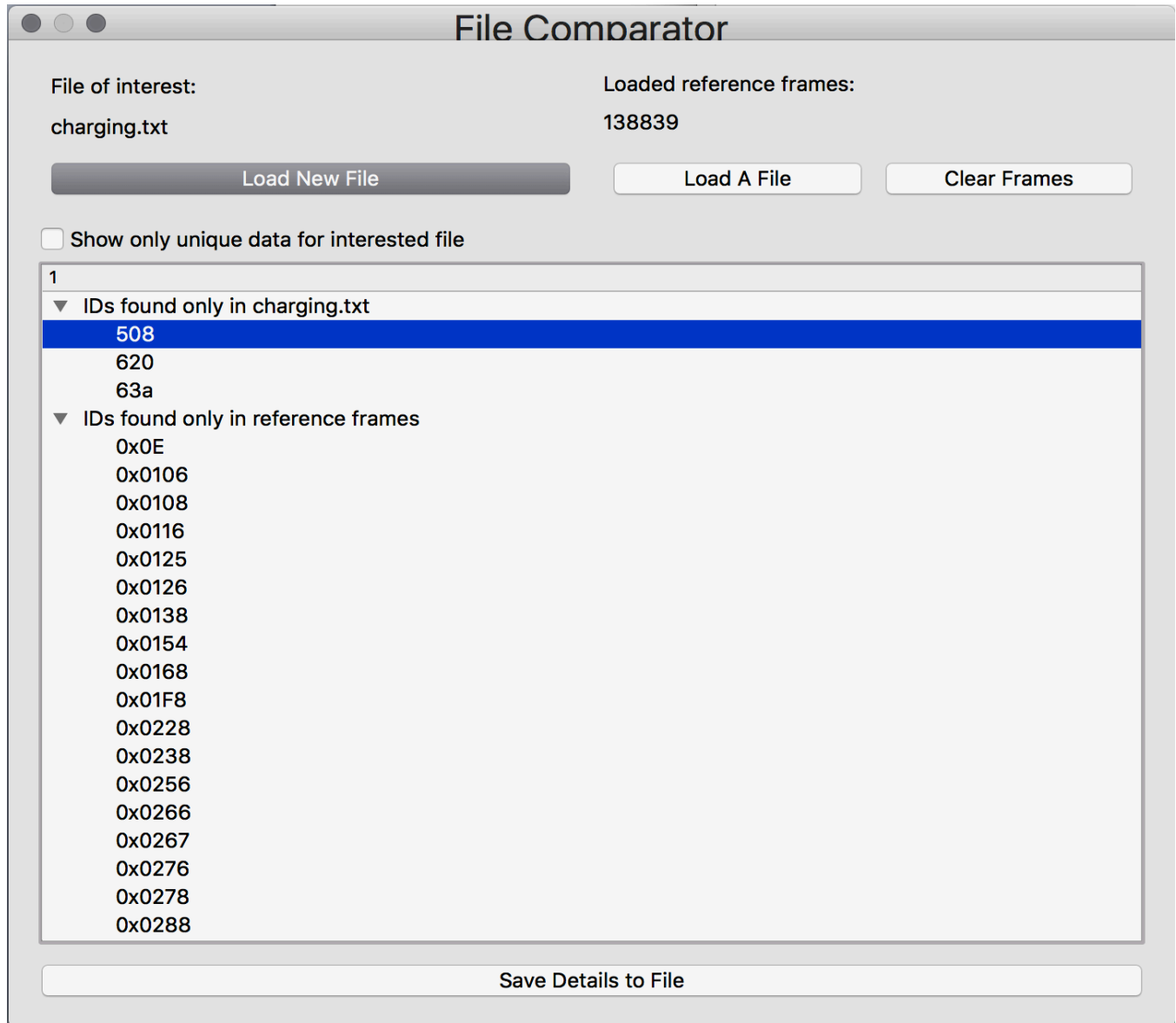
The screenshot shows a window titled "Detailed Frame Information" with two main panels. The left panel, "Frame IDs: (187 unique ids)", lists various CAN IDs, with 0x0116 highlighted. The right panel, "Details:", shows the following information for ID 0x0116:

- ID: 0x0116
- # of frames: 21903
- Data Length: 6
- Average inter-frame interval: 9999us
- Data Byte 0
- Data Byte 1
 - Range: 0x10 to 0xC0
 - Histogram:
 - 16/0x10: 307
 - 64/0x40: 20064
 - 144/0x90: 117
 - 160/0xa0: 86
 - 192/0xc0: 1329
- Data Byte 2
- Data Byte 3
- Data Byte 4
- Data Byte 5
- Bitfield Histogram:
 - 0 (Byte 0 Bit 0) :0
 - 1 (Byte 0 Bit 1) :0
 - 2 (Byte 0 Bit 2) :0
 - 3 (Byte 0 Bit 3) :0
 - 4 (Byte 0 Bit 4) :0
 - 5 (Byte 0 Bit 5) :0
 - 6 (Byte 0 Bit 6) :0
 - 7 (Byte 0 Bit 7) :0
 - 8 (Byte 1 Bit 0) :0
 - 9 (Byte 1 Bit 1) :0
 - 10 (Byte 1 Bit 2) :0
 - 11 (Byte 1 Bit 3) :0
 - 12 (Byte 1 Bit 4) :424
 - 13 (Byte 1 Bit 5) :86
 - 14 (Byte 1 Bit 6) :21393
 - 15 (Byte 1 Bit 7) :1532
 - 16 (Byte 2 Bit 0) :10757
 - 17 (Byte 2 Bit 1) :10763
 - 18 (Byte 2 Bit 2) :11006
 - 19 (Byte 2 Bit 3) :10552
 - 20 (Byte 2 Bit 4) :11774
 - 21 (Byte 2 Bit 5) :11512
 - 22 (Byte 2 Bit 6) :12049
 - 23 (Byte 2 Bit 7) :10025

FILE COMPARISON

File comparison is a powerful function allowing you to quickly compare two data logs and determine what message IDs they have in common, or conversely, which message IDs are unique to one or the other of the two files.

Lets assume we have done a CAN data capture of a car NOT charging and then the same car charging. We might want to see what new message IDs show up once we go into charge mode.



Our first order of business is to load our reference file. Click LOAD A FILE.

At this point LOADED REFERENCE FRAMES should give us an indication of the total number of reference frames in our file.

Click LOAD A NEW FILE and select a file to compare. The file name will be listed as the File of Interest.

At this point, we will see three expandable functions listed in the main screen.

IDs found only in file of interest.

IDS found only in reference frames.

IDS found in both.

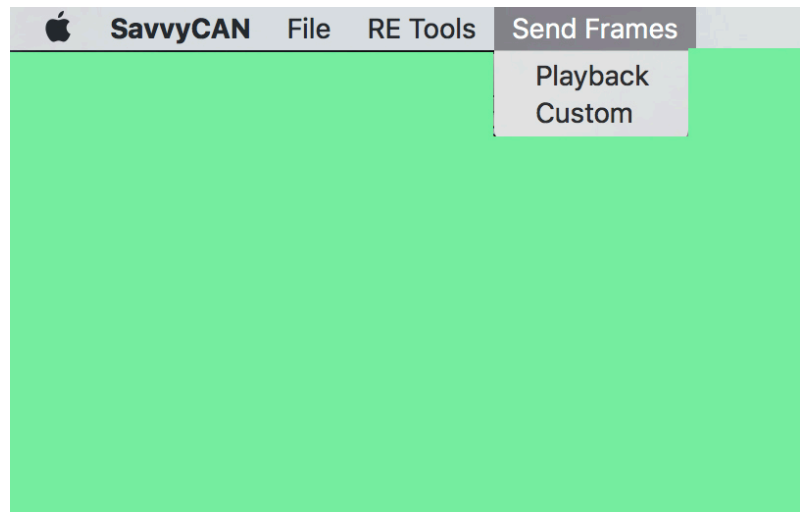
We can see that we have three new message IDs that appear in our charging.txt log that do not appear in our notcharging.txt reference file. They are 0x508, 0x620, and 0x63A. Those message IDs might bear further examination if we were seeking clues to the charging process in CAN data.

Finally SAVE DETAILS TO FILE will allow us to save this analysis to an external file for use elsewhere.

SENDING FRAMES – SIMULATING THE SYSTEM

In our work reverse engineering the CAN control of various OEM electric vehicle components, the central approach is to FIRST record an actual CAN bus message traffic of an operating CAR while it is in the act of doing what we want the component to do.

We can then play this recording back to a standalone component, and it should respond exactly the way it would if it were installed in the vehicle.



We basically then discard individual messages from the recording, and see if it will still do it.

Gradually we discard ALL the messages until we get down to JUST those messages required to get the component to do the task.

This allows us to focus our reverse engineering and analysis on JUST those messages necessary to operate the device – ignoring all others. For most devices, this is two or three messages but in some cases up to a dozen we send, and perhaps a similar number it responds with.

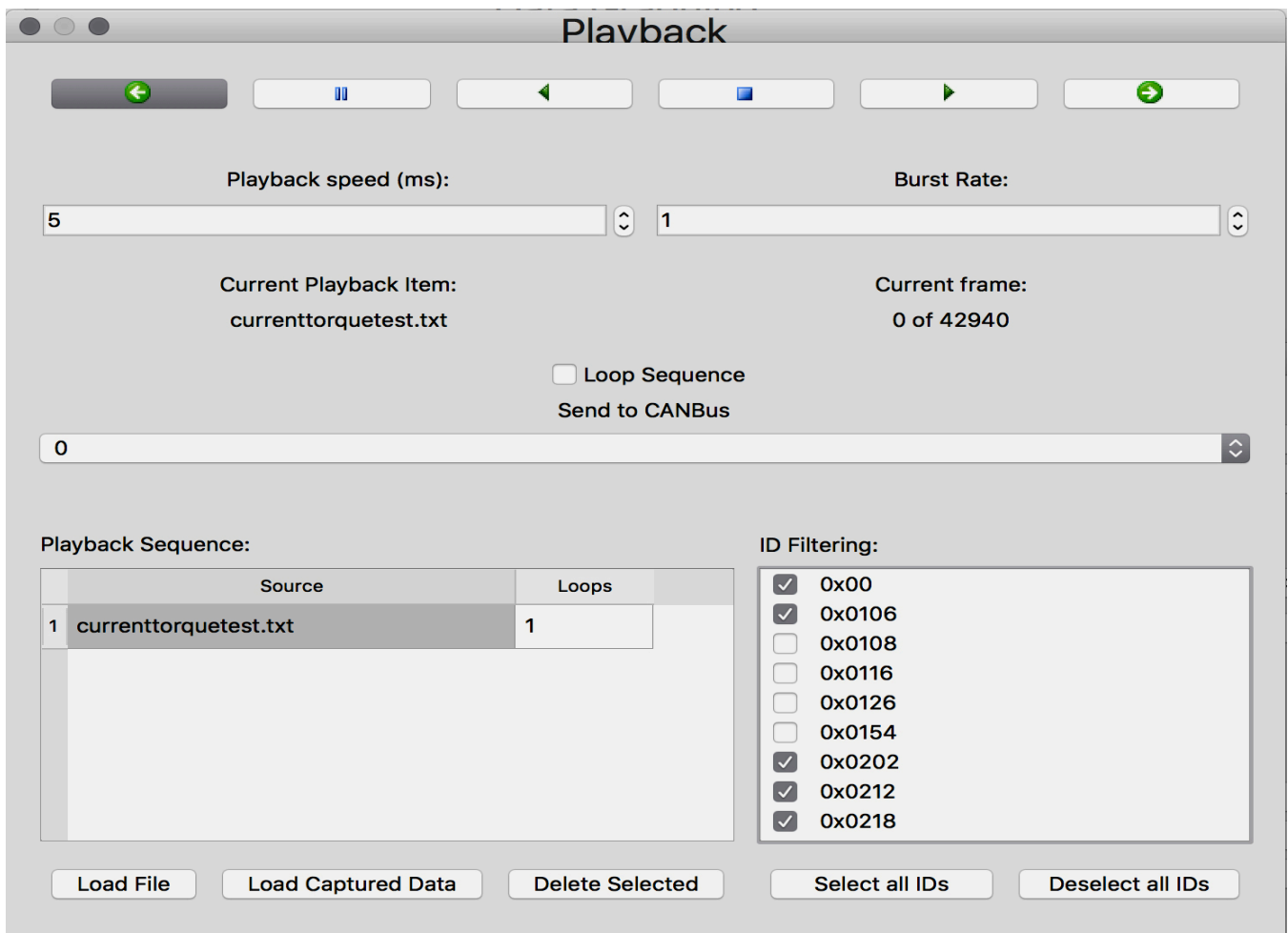
SavvyCAN provides two functions just for this. PLAYBACK and CUSTOM. PLAYBACK is oriented toward taking an entire log file, filtering some messages perhaps, and playing it back like a recording.

CUSTOM is more a function to design and send individual frames. You could for example, hookup SavvyCAN to an operating vehicle, and have it periodically transmit a single frame containing some data that changes the operation of the vehicle. It doesn't send an entire log file, just periodically inserts a message.







PLAYBACK

As described, PLAYBACK is provided to allow you to "play back" a previous data capture much as you would a tape recording. It simply puts the same messages back on a bus in the same order they were originally received.

The devil being in the details, it also allows you to modify this playback to some degree.



At the top of the playback screen are our playback controls – much like a video camera or tape player. From left to right,

-  – BACK steps backward one frame.
-  PAUSE – pauses the playback
-  REVERSE – plays backwards from current position
-  STOP/RESET – ends playback and returns us to frame one
-  PLAY – plays the current file forward
-  FORWARD ONE FRAME – sends the next frame in single step fashion

PLAYBACK SPEED AND BURST RATE

Playback speed and burst rate allow you to modify the speed at which frames are sent. Speed is normally a value in milliseconds between transmissions. Burst rate is more how many frames are sent during that transmission. Using these two variables, you can modulate the rate the CAN file is sent out on the bus. This can be quite important as timing is, as always, everything. It is quite common for equipment to receive frames every so many milliseconds. If they do not receive them before the timeout, they simply shut down and quit operating.

CURRENT PLAYBACK ITEM

This simply lists the file we are playing back.

CURRENT FRAME

This displays which frame we are on. For example 1215 of 42940 and gives us some idea of where we are at in the file. If for example, we know we did not put on the brake and put it in DRIVE until frame 2145 of the capture, we can detect that point in the process by observing this field.

LOOP SEQUENCE

This is a checkbox that determines what happens when we get to the end of the capture file. If it is unchecked, we simply stop. If it is checked, we immediately go to the first frame and resume transmission there. In this way, our file plays as a continuous loop.

SEND TO CAN BUS

This drop down menu allows us to select which CAN bus port the data goes out on,. One option is none.

PLAYBACK SEQUENCE

This allows actually quite complex playbacks using multiple files. And indeed it can do variable passes in those files. You could for example go through two iterations of file 1, a single iteration of file 2, and then three iterations of file three if desired.

ID FILTERING

The star of the playback screen is of course, again the ability to filter messages. This filter operates exactly as our display filter on the main capture screen, but in this case, applying to the messages transmitted onto the bus.

This is the most powerful function of SavvyCAN actually. To take a recording of CAN message traffic, transmit it onto a bus, and gradually decrease the messages sent until you only have the IMPORTANT ones for your task.

SELECT ALL IDS and DESELECT ALL IDS makes this easier. But again, the basics are that if you check a box, the message gets sent, if no checkbox, it does NOT get sent.

And note that you can do this WHILE a playback is occurring. So deselecting a message while transmitting a file simply discontinues sending that message going forward.

CUSTOM

Selecting CUSTOM from the top bar menu calls up the FRAME SENDER screen.

Frame sender allows you to quickly design and send custom CAN message frames onto the bus.

EN

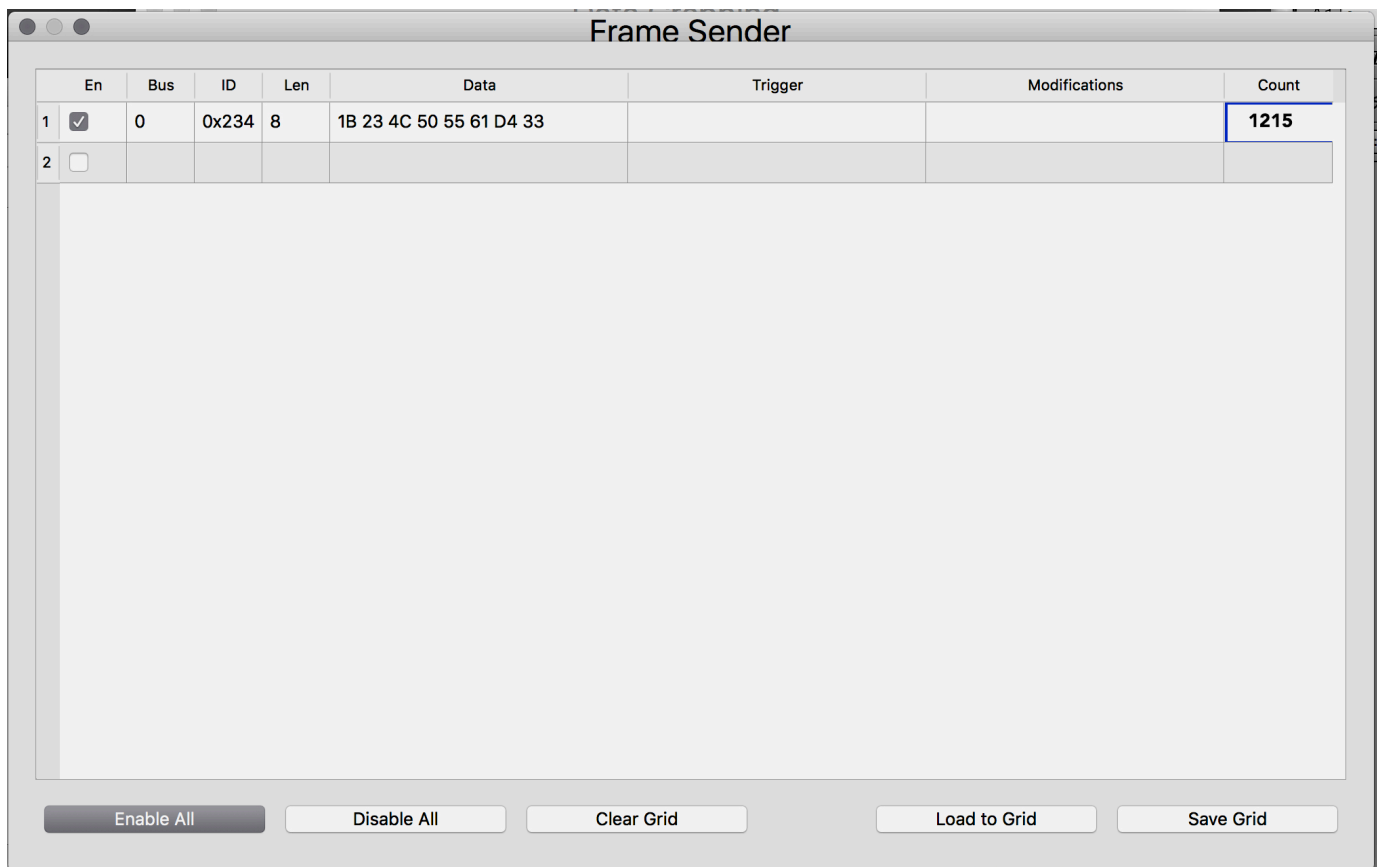
Enable. If this is checked, the program starts sending this message periodically. If you remove the checkbox, it discontinues.

BUS

Again, you can output the frame on any bus, typically 0 or 1.

ID

The message identification number of the message you want to simulate/send. Either 11-bit or 29-bit.



LEN

Length in bytes of data payload. Up to 8.

DATA

The data bytes you want to send.

TRIGGER

Triggers are a powerful and very flexible means to determine WHEN the described frame is sent out on the bus. And there are several conditions you can use to trigger this transmission.

ms

x

id

bus

ms

ms is the most basic form of trigger. It simply specifies that the defined CAN frame go out every x milliseconds. So **40ms** in this field would indicate to send the CAN frame every 40 milliseconds.

x

This is really a modifier to **ms**. It specifies the maximum number of times the message should be sent. So **40ms 100x** would indicate that the CAN frame should be sent every 40ms until 100 frames have been sent then terminate.

id

id is a trigger based on an INCOMING message frame. You can specify any message id and the described frame will be sent whenever that message is received.

This function also modifies the use of **ms** and **x**. If you specify an **id**, then **ms** is the time delay between when the message is received and when the described message is transmitted. **x** remains the maximum number of times this happens.

Id0x222 40ms 100x would cause the frame to be sent 40 milliseconds after 0x222 was received, but this would only occur on the first 100 such messages received.

bus

bus is a modifier for **id**. In this way, you can specify that not only the message ID has to be correct, but it has to come in on the correct bus.

id0x222 bus0 40ms 10x would then transmit 40 milliseconds after message ID 222 comes in on bus 0, but only for the first 10 instances. A 222 message incoming on bus1 would be ignored entirely.

These values can also be compounded in multiple triggers separated by commas:

id0x222 40ms, id0x111 10ms, 1000ms

This would transmit 40 milliseconds after a 222 message was received, but also 10 milliseconds after a 111 message was received. And in any case, it is going to transmit once every 1000 milliseconds (once per second) whether or not anything is received.

MODIFICATIONS

Modifications is again a powerful and flexible way of customizing our transmissions mathematically. Let's assume that the eight data bytes in our DATA section are numbered from 0 to 7 with the last data byte on the right being 7. In modifications, we will refer to these eight bytes as D0-D7.

Modifications allow us to perform mathematical operations on these eight data bytes. The operations include:

- + Addition**
- Subtraction**
- * Multiplication**
- / Division**
- & bitfield operation AND**
- | bitfield operation OR**
- ^ bitfield operation XOR**

So for example: **D0=D0*4** would replace the value in data byte 7 with that value times 4.

Or it could be a simple replacement: **D5=0xF3**. Or **D5=D3+0xF3**

These examples look more or less like nonsense. After all, we can put anything we like in the data fields anyway. But we can also use the value of incoming CAN message frames in our calculations and that makes it slightly more interesting. We do this with the **ID:** and **BUS:** commands.

D3=BUS:0:ID:0x222:D3

In this example, we are copying data byte 3 from any message 0x222 arriving on bus 0 into our own D3 data byte.

These can become quite exotic:

D3=BUS:0:ID:0x222:D3 + BUS:1:ID:0x123:D7 / 4

In this case, we take data byte 3 from message 222 arriving on bus 0, sum it with data byte 7 of message 123 arriving on bus1, and divide the result by 4. This calculated value is placed in our data byte 3 of the frame we are transmitting.

PUTTING TRIGGERS AND MODIFICATIONS TOGETHER

A simple example illustrates how powerful this can be. Message 222 on bus 0 carries motor coolant temperature in data byte four. This is on the bus going from the Vehicle Control Unit to an instrument cluster that displays it on a gage. We are going to SEND a message 0x222 of our own.

TRIGGER: **id0x222 bus0**

This indicates that we are going to send our frame immediately on receipt of a message ID 0x222 on bus 0.

MODIFICATIONS: D0=BUS:0:ID:0x222:D0, D1=BUS:0:ID:0x222:D1, D2=BUS:0:ID:0x222:D2, D3=BUS:0:ID:0x222:D3, D4=BUS:0:ID:0x222:D4 *2, D5=BUS:0:ID:0x222:D5, D6=BUS:0:ID:0x222:D6, D7=BUS:0:ID:0x222:D7

Although this is somewhat long, we are really copying all the data out of the received message 222 into our own replacement message 222 and sending it back out immediately. But in the process, coolant temperature in data byte 4 gets multiplied by 2 – essentially doubled.

The instrument cluster receives the message from the VCU, but every time it does, it immediately receives one from SavvyCAN doubling the value. And so the instrument cluster shows a coolant temperature twice as high as it actually is measured by the vehicle control unit.

THE GRID

You can enter actually a number of different messages on the screen in what we refer to as a GRID. Individual messages can be enabled or disabled using the EN field at different times and for different purposes.

At the bottom of the screen are several buttons for managing this grid.

ENABLE ALL

Enables all messages in the grid.

DISABLE ALL

Unchecks EN for all messages in the grid.

CLEAR GRID

Eliminates all current messages from grid.

SAVE GRID

Saves the current grid to an **.fsd** file for later use.

LOAD GRID

Loads grid from a previously saved **.fsd** file.

OBDII ADAPTER VERSION

A variant of the CAN adapter is the J1979 OBDII version. This version of the adapter features an ON/OFF power switch and an On Board Diagnostics version II (OBDII) female port much as found in almost all U.S. made vehicles since 2000.

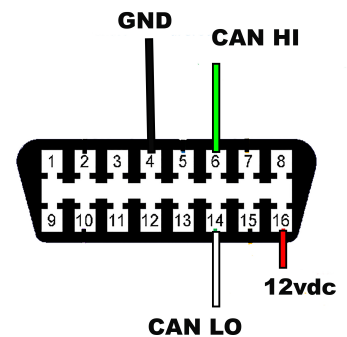
Please note that the OBDII version comes with preloaded software and uses Over The Air updates to update this software. If you install ESP_RET in its place, it cannot be reverted to OBDII.

OBDII is required by the Environmental Protection Agency since 1996 to enable emission testing of all manufactured vehicles in the United States and to standardize vehicle analysis for maintenance.

The Tesla Model 3 does not feature an OBDII connector at all. They were the first street legal electric vehicle to receive a waiver for this requirement.

Five signaling protocols are permitted with the OBD-II interface; most vehicles implement only one. It is often possible to deduce the protocol, based on which pins are present on the J1962 connector.

Today, by far the most common is the ISO 15765 [CAN](#) (250 kbit/s or 500 kbit/s). The CAN protocol was developed by Bosch for automotive and industrial control. Unlike other OBD protocols, variants are widely used outside of the automotive industry. While it did not meet the OBD-II requirements for U.S. vehicles prior to 2003, as of 2008 all vehicles sold in the US are required to implement CAN as one of their signaling protocols.



**Tesla Model 3 - pre January 2019
OBDII CAN Adapter**

In the years since the OBDII requirement, a number of third party products have evolved that take advantage of the data available on the OBDII port. Initially, this consisted of devices to clear trouble codes and diagnose vehicle faults. But it soon expanded to include Head Up Displays (HUD), specialized RPM gauges, and today includes monitoring equipment used by fleets and insurance companies to monitor driving.

The EVTV ESP32 CAN adapter, actually features TWO CAN ports. And so it can monitor the Vehicle CAN bus on the Tesla Model 3 with one port, and translate that data to J1979 protocol Parameter Identifiers (PIDs) on the other, while providing the actual standard OBDII J1962 connector on that port.

One of the most popular add-ons for OBDII is the ELM327 adapter. This originally translated OBDII PIDS to Universal Serial Bus (USB) to easily connect a laptop. But it further evolved to provide a wireless connection using WiFi, Serial Bluetooth, and eventually Bluetooth Low Energy (BLE_ – eliminating the necessity for the USB cable entirely.

THAT led to an entire series of add-ons using the ELM327 transmissions – including numerous smart phone and tablet applications.

The EVTV CAN adapter for the Tesla Model 3 uses the Espressif ESP32 microcontroller. That chip has wireless capability built in to the chip itself for WiFi, Bluetooth, and BLE.

EVTV has developed software to emulate an ELM327 WiFi module. You don't need to plug one into the OBDII port at all because the EVTV CAN adapter does it directly.

The most popular program for accessing this data via phone or tablet is called Torque Pro. It was written by Ian Hawkins and it runs on any Android platform device. It does NOT run on Apple iOS devices, but there are similar programs for Apple's iOS. However, Torque remains the most fully featured OBDII program to build a car "dashboard" available.

One of the reasons for its popularity is it has developed into a fascinating dashboard design kit. You can select from an amazing variety of dials and gages and data displays for all of the SAE standard PIDS, as well as color themes, backgrounds, etc. It has become a busy box for dashboard designs. All of these displays and gages can be connected to PIDS to display the data, and the user can specify the minimum and maximum values displayed, dial size and position, etc.

In this way, you can design your own interface to display any data you like from the OBDII standard PIDS. Additionally, Torque features additional PIDS derived from the phone and tablet itself, including GPS position, GPS speed, accelerometer data, and more.

More to our purposes, it supports the addition of customized PIDS.

And so EVTV has developed and tested a Torque interface for the Tesla Model 3. You are not limited to this at all. It illustrates the use of OBDII PIDS for electric vehicles. But you can design your own Torque dashboard using any of the PIDS available from the CAN adapter. And these include some specific custom PIDS developed for the Tesla Model 3.

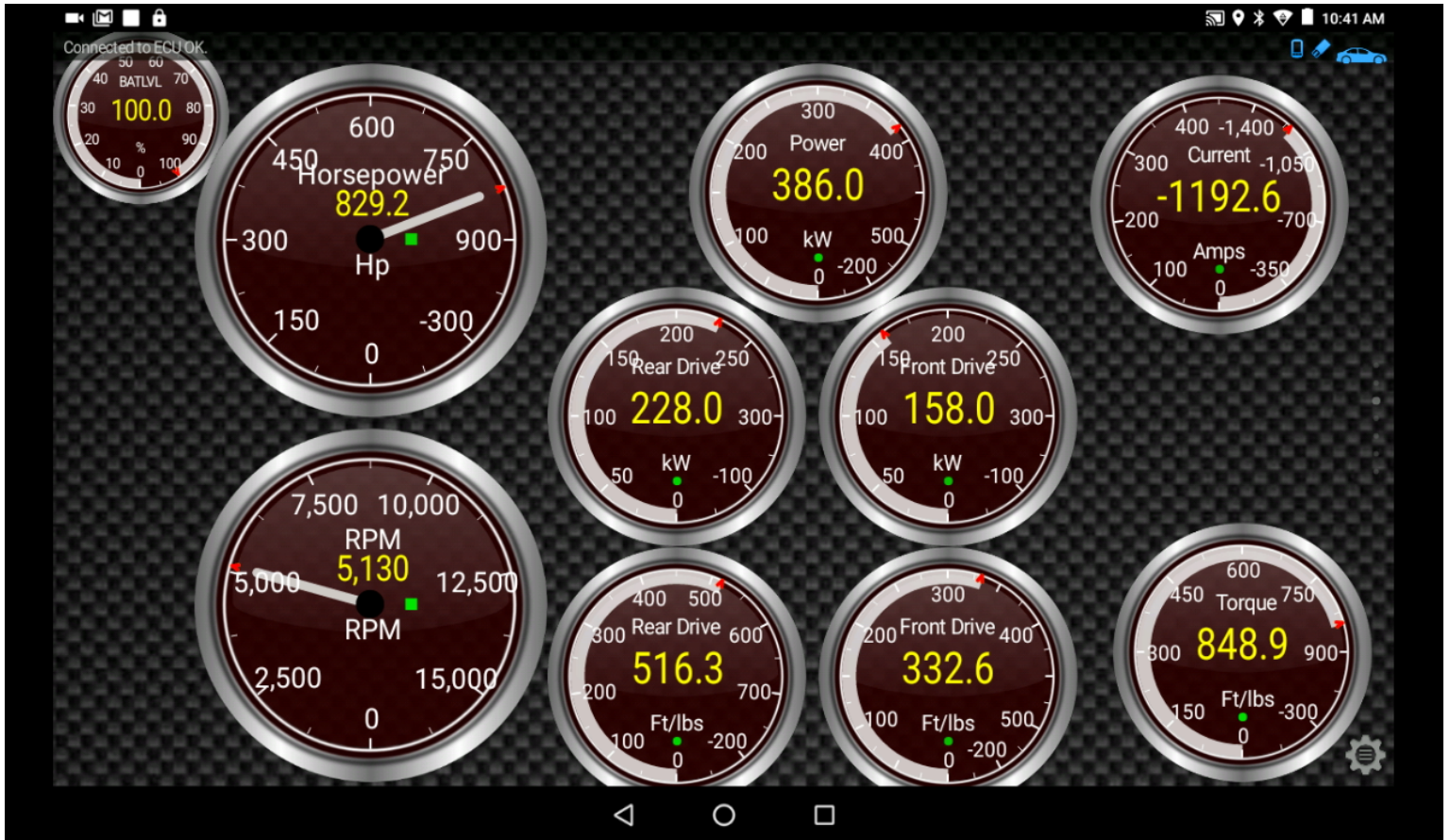
Electric vehicles simply have no corresponding data for many of the PIDS important to emission testing. We don't have Mass Air Flow and oxygen sensors and exhaust gas temperature and inlet air temperature. OBDII is all about emission testing of internal combustion engine vehicles.

But some obviously do – speed, rpm, etc. Fuel level can be an analog to our battery state of charge. But we have to be a bit more creative to find PIDs that can be repurposed to display battery voltage, power in kilowatts, inverter temperature, and so forth.

And so in this section, we describe the PID choices we made to try to translate a data stream intended for ICE cars to an electric vehicle and specifically the Tesla Model 3.

Note that we have translated a somewhat larger set of PIDs than we use in our example display. And that set of PIDS is subject to expansion as the need arises.





Because of this, we have also developed a means to update the adapter software using a wireless connection to your home wireless hub/router over the Internet. You can download and update the software very easily this way.

SAE J1979 PIDS

ISO 15765 describes the physical and electrical properties of the 500 kbps connection. But overlaying the basic CAN protocol is a higher level protocol for requesting and receiving diagnostic data. This is SAE J1979.

Basically it describes an initial connection and the transmission of a catalogue of parameter identifiers (PIDS) that the vehicle supports. The device connected to the J1962 port can then REQUEST any of these listed PIDS at any time. The vehicle will reply with the data described by that PID .

This was all designed for diagnostics and emission testing of internal combustion engine cars. Electric vehicles do not feature oxygen sensors, exhaust gas temperature, manifold air flow or temperature. Indeed MOST of the standard PIDS on ICE vehicles simply make no sense at all in an electric vehicle.

Throttle position and speed and RPM and the odometer obviously do. And we can “repurpose” some PIDS to carry information that IS of interest on an electric vehicle, such as battery voltage, state of charge, and inverter temperature.

The ESP32 OBDII adapter supports the following PIDS:

PID 0x04:

Calculated engine load ($A * 100 / 255$) – Percentage.

This is derived from the maximum torque value of the Model 3 compared to the current torque load.

PID 0x05:

Engine Coolant Temp ($A - 40$) = Degrees Centigrade

This PID derives from the **Model 3 motor inverter temperature**

PID 0x0A:

Fuel Pressure

We use fuel pressure as a proxy for the voltage of the **high voltage battery**

PID 0x0C:

Engine RPM ($A * 256 + B$) / 4

We use Engine RPM to represent **electric motor RPM** which is actually in this case rear axle RPM times 9 as the Tesla Model 3 currently has a 9:1 drivetrain gear reduction

PID 0x0D:

Vehicle Speed

If the Tesla Model 3 is currently in kilometers/hour mode, it sends vehicle speed as kilometers per hour. If in MPH mode, it sends as miles per hour.

PID 0x10:

MAF Air Flow as a proxy for motor horsepower. Torque has a display for horsepower it calculates from vehicle RPM and actual torque PIDs as well as reference torque for the vehicle.. But the adapter calculates it's own horsepower as well and sends as MAF air flow. Torque actual is in Newton Meters so it converts that to ft-lbs as well for the HP calculation.

horsepower=((model3.rearAxleRPM*9)*(model3.torqueActual*0.73756))/5252

PID 0x11:

Throttle position

This sends **Model 3 throttle position** as a percentage of full throttle

PID 0x1C:

Standard supported. Returns 1 = OBDII

PID 0x1F:**Runtime since engine start**

Return the actual board uptime in seconds. The adapter is powered by 12vdc from the vehicle which comes on when you enter the car typically. So this corresponds reasonably well.

PID 0x21:**Distance traveled with fault light lit ($A*256 + B$) - In km**

We use this distance PID to carry **Model 3 Odometer**.

PID 0x22:**Manifold Fuel Rail Pressure $0.079(A*256 + B)$**

We use this PID to carry High Voltage Battery Voltage. It is more accurate than Fuel Pressure PID 0x0A simply because it is a two byte value instead of a single byte. We can get a more accurate display this way.

PID 0x2F:**Fuel level** as a percentage of full

Model 3 SOC as a percentage. This value is the Battery Management System state of charge value. This is modified in the Model 3 for display. The Model 3 actually derives a UI SOC, a min SOC, a max SOC, and an avg SOC. Oddly, none match the actual SOC displayed on the center console. Basically they fudge the percentage a bit for the driver based on ambient temperature and other factors in the MCU. Minimum SOC seems closest.

PID 0x32:**Evaporative Air Pressure**

Used to carry **power in kilowatts** calculated by multiplying battery voltage and current.

PID 0x42:**Control Module Voltage**

Model 3 12v battery voltage.

PID 0x51:**Fuel type.**

Type 8 actually IS listed as battery electric for fuel in the PID standards.

PID 0x5B:**Hybrid battery pack remaining life**

Model 3 SOC as a percentage. This value is the Battery Management System state of charge value. This is modified in the Model 3 for display. The Model 3 actually derives a UI SOC, a min SOC, a max SOC, and an avg SOC. Oddly, none match the actual SOC displayed on the center console. Basically they fudge the percentage a bit for the driver for probably well intentioned though misguided purposes. Minimum SOC seems closest.

PID 0x61:

Driver requested torque as a percentage of full torque

PID 0x62:

Actual Torque delivered as a percentage of full torque

PID 0x63:

Reference torque for engine - presumably max torque in Newton Meters
Currently sent as 416 Nm

OBDDII ADAPTER CONFIGURATION

The OBDDII adapter version doesn't actually use the USB port for CAN traffic. But you can use USB to access a configuration screen that can be helpful.

Use any ASCII terminal program from any operating system on any computer. But set the communications settings for 115,200 bps, 8 data bits, no parity, 1 stop bit (8N1). And turn on line feeds and/or carriage returns to be sent with any command.

The screen will be blank until you enter "H" and press the return/enter key to send the command. Enter commands as shown with the value you want and a carriage return.

Build number: 109

System Menu:

Enable line endings (LF, CR, CRLF)

Short Commands:

h = help (displays this message)

R = reset to factory defaults

Config Commands (enter command=newvalue). Current values shown in parenthesis:

LOGLEVEL=1 - set log level (0=debug, 1=info, 2=warn, 3=error, 4=off)

CAN0EN=1 - Enable/Disable CAN0 (Vehicle CAN bus 0 = Disable, 1 = Enable)

CAN0SPEED=500000 - Set speed of CAN0 in bits per second (125000, 250000, etc)

CAN0LISTENONLY=0 - Enable/Disable Listen Only Mode (0 = Dis, 1 = En)

CAN1EN=1 - Enable/Disable CAN1 (OBDDII Port 0 = Disable, 1 = Enable)

CAN1SPEED=500000 - Set speed of CAN1 in bits per second (125000, 250000, etc)

CAN1LISTENONLY=0 - Enable/Disable Listen Only Mode (0 = Dis, 1 = En)

FORWARDCAN=0 - Forward ALL CAN traffic from CAN0 to CAN1 (0 = No, 1 = Yes)

SSID=ESP320BD2 - SSID for creating an adapter access point

WPA2KEY= - WPA2 key to use for adapter access point

WIFIMODE=1 - 0 = Connect to a local AP as client, 1 = Make an adapter AP

CLIENTSSID=riverhouse - SSID of local AP

CLIENTWPA2KEY=usatoday - WPA2 key to use when connecting to local AP

ALLOWSC= 0 - Allow SavvyCAN WiFi connections (0 = no, 1 = yes)

UPDATE - Update software automatically from EVTV server (Requires SSID and key of local AP with Internet)

BUILD NUMBER

This is the version number of the software running on the adapter. Useful as you may be updating it using the automatic update function.

LOG LEVEL=1

We use this for debugging mostly. Leave at 1.

CAN0EN=1

This is the CAN bus 0 enable function. CAN 0 is connected to the Model 3 adapter and scans the Model 3 Vehicle Drive bus for CAN messages from the vehicle. Set to 1 to enable it.

CAN0SPEED=500000

This is the CAN speed for CAN0. Since the vehicle bus is always 500,000 you probably do not want to change this.

CAN0LISTENONLY=0

We can set a CAN channel where it “listens only” and does not send any acknowledgements or other signals on the bus.

CAN1EN=1

This is the CAN bus 1 enable function. CAN 1 is connected to the J1962 On Board Diagnostics Connector. Set to 1 to enable it.

CAN1SPEED=500000

This is the CAN speed for CAN1. Normally set to 500,000 but some OBDII devices require 250000.

CAN1LISTENONLY=0

We can set a CAN channel where it “listens only” and does not send any acknowledgements or other signals on the bus. This doesn't make much sense on the OBDII connectors.

FORWARDCAN=0

We normally observe CAN0 traffic and use the information to respond to PID requests on CAN1. But some applications require access to ALL CAN traffic. By setting FORWARDCAN to 1 you enable this feature and the adapter will actually act as a bridge, copying all incoming CAN0 frames out CAN1 to the OBDII connector.

But note that the Model 3 sends CAN data at rates up to 2000 frames per second. This CAN simply overwhelm many OBDII devices and normal PID requests are set to a much lower priority by using a very high message ID number. So only forward if you really need to do so as it can make Head Up Displays and phone/tablet applications much slower.

On the other hand, the EVTV OBDII adapter uses WiFi to transmit data. If your application requires Bluetooth Low Energy for example, you can plug an ELM 327 BLE adapter into the OBDII connector and receive all data over the OBDII connection via the ELM327 bluetooth signal.

SSID=ESP32OBD2

This can be any short name you like. This how you specify Station Set Identifier (SSID) of the wireless access point set up by the adapter. You would then “connect” to this AP from your laptop or tablet or smart phone.

WPA2KEY= -

Currently set to – for NO password. In this way, devices can easily connect to the SSID without entering a password. For security, you may want to add one.

WIFIMODE=1

This is for all intents and purposes deprecated. You will almost always want to operate in mode 1 as this allows the adapter to BE a wireless access point you can connect to.

In theory, you could set it to connect to your local hub, and set your device to connect to the same hub to receive the data. This doesn't make much sense in a car.

CLIENTSSID=EVTVHUB

This is how you specify the local access point that has an internet connection to connect the adapter to primarily to obtain software updates.

CLIENTWPA2KEY= EvTv

This is the password needed to access your local AP to connect to the Internet.

ALLOWSC= 0

The OBDII version of this adapter does NOT allow SavvyCAN to access data via USB port at all. But it can do so from a wireless capable PC. And it can also access it from the J1962 adapter.

This variable determines whether you allow the SavvyCAN program to monitor CAN traffic wirelessly from the adapter. Note that SavvyCAN can not only capture data this way, but it can play back recorded data IN to this adapter in this way and it will appear on the vehicle data bus. Loads of fun. BUT if SavvyCAN can do this, potentially others could transmit data into this port and similarly have fun putting CAN commands onto the Vehicle CAN bus and controlling your car.

No amount of security would be appropriate here. But by setting this to ZERO, you ensure the adapter doesn't receive anything at all that could be put on the bus.

Note that the facility to forward data from the vehicle bus to the OBDII port is a one way trip. NO data from the physical port OR received wirelessly can go the other way BACK onto the vehicle bus with the lone exception of SavvyCAN and then only when this is set to 1. Normal operation should always have this set to 0.

Set to 1 to enable a SavvyCAN connection for capture and analysis. Don't forget to set this back to 0 when that analysis/test is complete as leaving it enabled poses a severe security risk to your car.

You can connect SavvyCAN via the OBDII J1962 connector. This requires BOTH the FORWARDCAN=1 and the ALLOWSC=1 to work. And you will need a second CAN adapter wired to connect to the J1962 mating connector.

UPDATE

UPDATE is one of the more interesting features of the OBDII adapter. IF you have a client SSID and password entered above and if your local wireless hub is up and connected to the Internet, the **UPDATE** command will cause the adapter to turn off it's AP, connect to your Wifi AP, and go out through the internet to download the latest binary version of this software. It will then reboot the adapter into this latest software version.

In the event you don't LIKE the new version, enter **REVERT** to go back to the previous version in memory.

TORQUEPRO Tesla Model 3 Dashboard

TorquePro is an Android application allowing you to design your own vehicle "dashboard" and display it. Each gage is tied to a parameter identifier (PID) transmitted by the ESP32 OBDII adapter.

In this way you can "view" your Tesla Model 3 data in real time.

You can select from a variety of gages and dials and graphs in designing your dashboard. And you can save and load different dashboards from memory.

To get started, we have created the TeslaModel3 dash file to demonstrate what is possible with TorquePro. You can modify and extend this file, or start over entirely with your own design.

The file is titled **TeslaModel3.dash** and may be downloaded from the ESP32 OBDII description on our web site at <http://store.evtv.me>.

To use it:

1. Place the file in the **InternalStorage/.torque/dashboards** directory.
2. Start TorquePro and select **REALTIME INFORMATION**.
3. Tap the settings GEAR symbol to bring up settings.
4. Select **Layout Settings**
5. Select **Import Layout**
6. Select **TeslaModel3.dash**

TORQUEPRO Tesla Model 3 Extended PIDs

TorquePro provides a means for adding extended or modified Parameter Identifiers – basically custom PIDS.

The PIDS are provided in a Comma Separated Values – CSV file. It must follow a specific Torque app format and appear in the [.torque/extendedpids](#) folder. You must load it using the MANAGE PIDS function one time. Thereafter it will always be part of your vehicle profile.

We have created a [TeslaModel3.csv](#) file for just this purpose. To use it:

1. Download the TeslaModel3.csv file from the detailed description of the OBDII adapter on the web site <http://store.evtv.me> using your Android device.
2. Turn on **SHOW HIDDEN FILES** in your Android file manager.
3. Copy the file into the hidden directory **InternalStorage/.torque/extendedpids**
4. Bring up the Torque Application main screen.
5. Tap on the gear icon to select settings and then tap **SETTINGS**.
6. Scroll down until **Manage extra PIDS/Sensors** appears on menu and select it.
7. Press the three vertical dots in the upper right hand of screen to bring up a menu.
8. Select **Add predefined set** from the menu.
9. Select **TeslaModel3** from the list that appears.

Once you have imported the **TeslaModel3.csv** file, you need not do so again. The PID extensions will remain in your vehicle profile each time you load the app.

Subsequently, when you add a gauge to your dashboard, the Tesla PIDS will appear at the top of the list of PIDs to select from. We prefaced each entry with a period so they appear first in the lengthy list of PIDs provided by Torque.

Generally, the OBDII adapter supports standard PIDS where they make sense. But the additional Tesla specific definitions are often more accurate and useful. And in some cases, they represent values that cannot be carried in standard PIDS. Both will continue to work.

TeslaModel3.csv

```
"Name", "ShortName", "ModeAndPID", "Equation", "Min Value", "Max Value", "Units", "Header",
"startDiagnostic", "stopDiagnostic"

".Tesla Battery Voltage", "Battery", "019A", "FLOAT32(A:B:C:D)", "280", "400", "VDC", "7e0", "", ""
".Tesla RPM", "RPM", "019B", "(A*256+B)", "0", "15000", "RPM", "7e0", "", ""
".Tesla Motor Temp", "Stator", "019C", "A", "0", "100", "C", "", "", ""
".Tesla KWH Remaining", "Remaining", "019D", "(A*256+B)/100", "0", "100", "kWh", "7e0", "", ""
".Tesla Rear Torque NM", "Rear Drive", "01A1", "FLOAT32(A:B:C:D)", "-60", "360", "Nm", "", "", ""
".Tesla Front Torque NM", "Front Drive", "01A2", "FLOAT32(A:B:C:D)", "-60", "360", "Nm", "", "", ""
".Tesla Rear Torque Ft/lbs", "Rear Drive", "01A1", "FLOAT32(A:B:C:D)*0.73756", "-60", "360", "Ft/lbs", "", "", ""
".Tesla Front Torque Ft/lbs", "Front Drive", "01A2", "FLOAT32(A:B:C:D)*0.73756", "-60", "360", "Ft/lbs", "", "", ""
".Tesla Battery Current", "Current", "01A3", "FLOAT32(A:B:C:D)", "-1000", "500", "Amps", "7e0", "", ""
".Tesla Power", "Power", "01A4", "FLOAT32(A:B:C:D)", "-100", "300", "kW", "7e0", "", ""
".Tesla Horsepower", "Horsepower", "01A5", "FLOAT32(A:B:C:D)", "0", "300", "Hp", "7e0", "", ""
".Tesla Torque", "Torque", "01A6", "FLOAT32(A:B:C:D)", "0", "500", "Nm", "7e0", "", ""
".Tesla Torque Ft/lbs", "Torque", "01A6", "FLOAT32(A:B:C:D)*0.73756", "0", "300", "Ft/lbs", "7e0", "", ""
".Tesla State of Charge", "SOC", "01A7", "FLOAT32(A:B:C:D)", "0", "100", "%", "7e0", "", ""
".Tesla Rear Drive Unit Power", "Rear Drive", "01A8", "FLOAT32(A:B:C:D)", "-100", "300", "kW", "7e0", "", ""
".Tesla Front Drive Unit Power", "Front Drive", "01A9", "FLOAT32(A:B:C:D)", "-100", "300", "kW", "7e0", "", ""
```

Name

The name is a title that will appear in the PID list for TorquePro when you add a gage. We preface the names with a period so they will appear at the top of the lengthy list of PIDs TorquePro supports.

Short Name.

The Short Name is actually text that will appear top center of most gages you might use identifying what is measured.

Mode and PID

All custom PIDs used by the Tesla Model 3 OBDII adapter will use mode 01 as specified by the first two characters of this field. It is the most common PID mode.

The subsequent two characters are the hexadecimal address of the specific PID described.

Equation

The Equation is an arithmetic operation performed by TorquePro on the data received from this PID. Normally A is the first data byte of the transmission, B second, C third, etc. PIDs may be as short as one byte or much longer in some cases.

For maximum precision and accuracy, we often store 32-bit floating point variables in a four byte PID for transmission and TorquePro can then reconstruct these directly.

Min Value

The minimum value is the minimum data that will be shown on the display. You can easily modify this when adding a display, but this represents the default value.

Max Value

Maximum value represents the maximum value displayed by a gage. Again this is the default and can be overridden later during gage design. And feel free to do so. Because of the variety of Tesla Model 3 configurations, you WILL find the maximum value for an all-wheel-drive “performance” version completely different from a rear wheel drive standard version for example. TorquePro makes this easy to accomplish.

Units

Units is a simply short text string that will be shown bottom center on most gages to define what is being measured – kW, VDC, Amps, Mph, etc.

Header

Header is the type of request response related to mode. If left empty it will be automatic. But we set to 7E0 for clarity. This is the most common PID request mode.

SAVVYCAN and OBDII Version

As previously noted, the OBDII Version software is NOT open source and is normally updated by a binary over the air wireless update from our Amazon S3 bucket storage.

For some, the advantages of open source and USB connection to the adapter may be of advantage and we make this adapter available with the ESP32_RET reverse engineering tool software separately. Generally you will find high speed performance for data logging slightly better using the ESP_RET version as it is a smaller, lighter, faster program.

But the OBDII version has some SavvyCAN connectivity and indeed it makes wireless data logging very easy. It CANNOT be used with USB connection.

To use it with SavvyCAN, on the OBDII configuration screen you must set ALLOWSC=1.

To “connect” with the adapter from SavvyCAN running on a laptop:

1. Turn on the ESP32 OBDII adapter by activating the ON/OFF switch. The car must be active to provide 12vdc to power the device.

2. Use the normal laptop wireless feature to “connect” to **ESP32OBD2** access point. Note that this SSID can be changed in the configuration program for the adapter and a password may be optionally specified.
3. Select the SavvyCAN application.
4. Select **CONNECTION** to bring up the connection window.
5. Select **ADD NEW DEVICE CONNECTION** to bring up the **NEW CONNECTION** window.
6. Select **NETWORK** as the connection type.
7. Select **192.168.0.10** from the picklist displayed.
8. Select **CREATE NEW CONNECTION**.

This should result in a connection being shown on the connection window. It should indicate **CONNECTED** or **NOT CONNECTED**. If it is **NOT CONNECTED**, you will need to repeat the procedure to get an indication of **CONNECTED**.

At that point, you should see CAN data streaming in the main SavvyCAN window. Set **AUTO SCROLL WINDOW** to observe this.

After completing your logging session, save the data collected as follows:

1. On SavvyCAN top menu bar select **FILE**.
2. On drop down menu list select **SAVE LOG FILE**.
3. You should get a standard file saving window depending on your laptop operating system allowing you to name and save the log file. This can be done in a variety of formats.

It is also possible to connect SavvyCAN to the J1962 OBDII connector by using a second CAN adapter for SavvyCAN. It must be wired for the correct pins on the mating J1962 connector. Additionally, two settings are required:

FORWARDCAN=1 This echoes all vehicle CAN to the J1962 connector.

ALLOWSC=1 This allows all CAN received on the J1962 adapter to be echoed back to the vehicle CAN bus.