Tesla Model S CAN Bus Deciphering

Jason Hughes

(wk057)

Initial Public Release
v0.1 - 2016-01-15

DOCUMENT IS NOT OFFICIAL OR AFFILIATED WITH TESLA MOTORS. USE AT OWN RISK.

Little bit of a mess, but the info is here! :)

CAN3, ID 0x0102

- Description: BMS Current and Voltage
- Lengths observed: 6 and 8 bytes
- Frequency on bus: Approximately 100 Hz
- Example frames:
    - 0x66 0x97 0xF8 0xA6 0xF2 0x4D 0x28 0x01
    - 0x5C 0x99 0x9F 0x83 0x3C 0x07
- Note: Single motor cars appear to use 6-byte frames, dual motor cars 8-byte frames
    - Limited data set. This could be incorrect.
- Value descriptions
    - Bytes 1/0
        - Unsigned 16-bit value representing battery pack voltage
            - Low byte in byte 0, high byte in byte 1
        - Unit: 0.01V
        - Example: 0x66 0x97 = 387.58V
    - Bytes 3/2
        - Signed **15-bit** representing pack current
            - Low byte in byte 2, high byte in byte 3
            - high bit (bit 7) of byte 3 is unknown
            - bit 6 of byte 3 is the sign bit
        - Unit: 0.1A
        - Offset: Varies
            - In the dual motor data there is an offset of 10000 units
                - Subtract 10000 from the value to get the value in the above units.
            - No offset for single motor data
            - This derived from a limited dataset
        - Example (no offset): 0x9F 0x83 = 92.7A (regen)
        - Example (no offset): 0xD1 0xD2 = -1,156.7 A (consumption)
        - Example (-10000 offset): 0x85 0xA8 = 37.3 A (regen)
        - Example (-10000 offset): 0x27 0xF4 = -1,303.3 A (consumption)
    - Bytes 5/4
        - Unused by the instrument cluster (no change through whole range of values)
        - Closely matches bytes 3/2 at double the scaling, but not perfectly
        - Update: This is “unfiltered” current
    - Byte 6 *(8-byte frame only)*
        - Negative terminal temperature degrees C
        - temp_C = ((byte6 + (byte7 & 0x07)<<8)) * 0.1

CAN3, ID 0x0256
(Part 1 of 2)

- Description: Speed and cruise control status (Rear drive unit status)
- Length observed: 8 bytes
- Frequency on bus: Approximately 10 Hz
- Example frames:
  - 0x84 0x19 0x40 0x31 0x00 0xF2 0x20 0x78
  - 0x84 0x28 0xC0 0x31 0x5A 0x52 0x2D 0xCE
- Value descriptions
  - Byte 0 bit0-2
    - Drive inverter system state
    - 0 = Unavailable; 1 = Standby; 2 = Fault; 3 = Abort; 4 = Enabled; 5-7 = Undefined
  - Byte 0 bit3-5
    - Vehicle Hold state
    - 0 = Unavailable; 1 = Standby; 2 = Blend In; 3 = StandStill; 4 = Blend Out; 5 = Park; 6 = Fault ; 7 = Init
  - Byte 0 bit6 - Drive Inverter Proximity?
  - Byte 0 bit7 - Inverter ready
  - Byte 1
    - Bit 0
      - Regen light
    - Bits 4-7 (high nibble)
      - Cruise control status
      - 0 = OFF, 1= STANDBY, 2= ENABLED, 3= STANDSTILL, 4=OVERRIDE; 5=FAULT; 6=PRE_FAULT; 7=PRE_CANCEL
  - Bytes 3 bits 0-3/2 bits0-7
    - Unsigned **12-bit** value representing speed in MPH
    - Unit: 0.1 MPH
    - Note: Used by the pre-autopilot IC to display the analog speed bar value
    - Note: Always sent as MPH and displays correctly on the IC's analog speed bar regardless of the user's chosen unit (MPH or KPH)?
  - Byte 3 bit 7
    - Speed unit setting?
      - 0 = MPH, 1 = KPH
  - Byte 4, bit0 of Byte 5
    - Unsigned **9-bit** value representing cruise control speed setting
      - Byte 4 is low byte, bit0 of Byte 5 is the high bit of the 9-bit value
    - Unit: 0.5 MPH
    - Note: Used by the IC to indicate the user's chosen cruise speed
    - Note: Always sent as MPH and displays correctly on the IC regardless of the user's chosen unit (MPH or KPH)?
  - Byte 5
    - High nibble: counter
    - Bit 0 (see above on Byte 4)
    - Bits 1-3 - Something to do with emergency braking?

CAN3, ID 0x0256
(Part 2 of 2)

- Byte 6
  - Unsigned 8-bit value representing the numeric speed to display
  - Unit: 1 MPH **or** 1 KM/H
  - Note: This value is displayed on the IC as the vehicle's speed as the numeric/digital read out.
  - Note: This is the value displayed regardless of the user's chosen unit
    - Seems to depend on byte3 bit 7 (see above)
- Byte 7
  - Some kind of checksum...

CAN3, ID 0x0106

- Description: Rear drive unit info
- Length observed: 8 bytes
- Frequency on bus: Approximately 100 Hz
- Example frames: 31 FF 2D 7F A3 14 0E A8 - motorRPM: 5283 RPM - pedalPos: 5.6%
- Value descriptions
  - motorRPM = (byte4 + (byte5<<8)) - (512 * (byte5&0x80))
    - No scaling offset or sign
    - 16-bit signed
  - pedalPos = byte6 * 0.4
    - 0% to 102%
    - Max observed is 100%

CAN3, ID 0x01D4

- Description: Front drive unit measurement info
- Length observed: 8 bytes
- Frequency on bus: Approximately 100 Hz
- Example frames: 00 03 00 00 08 62 E1 23 - torqueMeasured: 88.5 Nm (65.27 ft/lb)
- Value descriptions
  - front_torque_Measured = (byte5 + ((byte6 & 0x1F)<<8) - (512 * (byte6 & 0x10))) * 0.25
  - Value is in Nm, signed 13-bit
    - Convert to ftlb with * 0.737562149

CAN3, ID 0x0154

- Description: Rear drive unit measurement info
- Length observed: 8 bytes
- Frequency on bus: Approximately 100 Hz
- Example frames: 30 63 FA FA 48 CC 09 F9
    - torqueMeasured: 627 Nm (462.45 ft/lb)
    - pedalPosA: 100%
    - pedalPosB: 100%
- Value descriptions
    - rear_torque_Measured = (byte5 + ((byte6 & 0x1F)<<8) - (512 * (byte6 & 0x10))) * 0.25
    - Value is in Nm, signed 13-bit
        - Convert to ftlb with * 0.737562149
    - pedal_position_sensor_A = byte2 * 0.4
    - pedal_position_sensor_B = byte3 * 0.4
        - 0% to 102%
        - Max observed is 100%
        - Redundant potentiometers in pedal

CAN3, ID 0x0266

- Description: Rear drive unit power info
- Length observed: 8 bytes
- Frequency on bus: Approximately 10 Hz
- Example frames: 85 B1 9F 01 3E EB 8A 44
    - inverter12V: 13.3 V
    - dissipation: 22125 W
    - drivePowerMax: 349 kW
    - mechPower: 207.5 kW
    - statorCurrent: 830 A
    - regenPowerMax: 72 kW
- Value descriptions
    - inverter12V = byte0 / 10
        - Volts
    - dissipation = byte1 * 125
        - Watts
    - mechPower = ((byte2 + ((byte3 & 0x7)<<8)) - (512 * (byte3 & 0x4))) / 2
        - kW (11-bit signed, 0.5 kW scale)
    - statorCurrent = byte4 + ((byte5 & 0x7)<<8)
        - A, no scaling or offset, 11-bit unsigned
    - regenPowerMax = (byte7 * 4) - 200
        - kW, 8-bit, scale 4, offset of -200
        - Weird that this is offset by -200… not sure how max regen can be negative…
    - drivePowerMax = (((byte6 & 0x3F)<<5) + ((byte5 & 0xF0)>>3)) + 1
        - kW
        - lowest value is 1 kW

CAN3, ID 0x02E5

- Description: Front drive unit power info
- Length observed: 8 bytes
- Frequency on bus: Approximately 10 Hz
- Example frames: 88 62 08 01 D9 99 07 00
    - inverter12V: 13.6 V
    - dissipation: 12250 W
    - drivePowerMax: 243 kW
    - mechPower: 132 kW
    - statorCurrent: 473 A
- Value descriptions: ***Same as rear drive unit, 0x0266 with the exception of no regenPowerMax***

CAN3, ID 0x0145

- Description: Front drive unit torque info
- Length observed: 3 bytes
- Frequency on bus: Approximately 100 Hz
- Example frames: 65 C0 6B - torqueEstimate: 50.5 Nm (37.25 ft/lb)
- Value descriptions:
    - torqueEstimate = ((byte0 + ((byte1 & 0xF)<<8)) - (512 * (byte1 & 0x8))) / 2
    - Value is in Nm, signed 12-bit
        - Convert to ftlb with * 0.737562149

CAN3, ID 0x0116

- Description: Rear drive unit torque and statusinfo
- Length observed: 6 bytes
- Frequency on bus: Approximately 100 Hz
- Example frames: 5A 44 75 44 8E FC
    - gear: 4 (D)
    - gearRequest: 4 (D)
    - torqueEstimate: 557 Nm (410.82 ft/lb)
    - vehicleSpeed: 32.05 MPH
- Value descriptions:
    - torqueEstimate = ((byte0 + ((byte1 & 0xF)<<8)) - (512 * (byte1 & 0x8))) / 2
    - Value is in Nm, signed 12-bit
        - Convert to ftlb with * 0.737562149
    - vehicleSpeed = ((byte2 + ((byte3 & 0xF)<<8)) - 500) / 20
        - MPH, 0.05 MPH resolution
    - gearRequest = (byte3 & 0x70)>>4
    - gear = (byte1 & 0x70)>>4
        - For both of the above: 0,5,6,7 = Invalid; 1=Park;2=Reverse;3=Neutral;4=Drive

CAN3, ID 0x0232

- Description: Battery Power limits
- Length observed: 4 bytes
- Frequency on bus: Approximately 10 Hz
- Example frames: 80 19 10 89
    - maxDischargePower: 350.88 kW
    - maxRegenPower: 65.28 kW
- Value descriptions:
    - maxDischargePower = (byte2 + (byte3<<8))/100
    - maxRegenPower = (byte0 + (byte1<<8))/100
    - Both result in a value in kW

CAN3, ID 0x0562

- Description: Battery Odometer
- Length observed: 4 bytes
- Frequency on bus: Approximately 0.1 Hz
- Example frames: 3C 27 2C 01 - Battery_odometer: 19670.844 miles
- Value descriptions:
    - Battery_odometer = (byte0 + (byte1<<8) + (byte2<<16) + (byte3<<24))/1000;
    - 32-bit unsigned value, 0.001 mile resolution
    - This is the BATTERY mileage, not necessarily the same as the car mileage

CAN3, ID 0x03D2

- Description: Battery Lifetime Energy Stats
- Length observed: 8 bytes
- Frequency on bus: Approximately 1 Hz
- Example frames: 25 AE 86 00 07 DC 8F 00
    - kwhChargeTotal: 9427.975 kWh
    - kwhDischargeTotal: 8826.405 kWh
- Value descriptions:
    - kwhDischargeTotal = (byte0 + (byte1<<8) + (byte2<<16) + (byte3<<24))/1000
    - kwhChargeTotal = (byte4 + (byte5<<8) + (byte6<<16) + (byte7<<24))/1000
    - Value for both using above results in kWh
    - Two 32-bit unsigned values with 0.001 kWh scaling (basically Wh scaling)
    - Note: Appears to take into account charging losses of some kind?

CAN3, ID 0x0302

- Description: Battery State of Charge (SoC)
- Length observed: 3 bytes
- Frequency on bus: Approximately 1 Hz
- Example frames: 82 DF 0D ; socMin: 89.8% ; socUI: 88.7%
- Value descriptions:
    - socMin = (byte0 + ((byte1 & 0x3)<<8)) / 10
    - socUI = ((byte1>>2) + ((byte2 & 0xF)<<6)) / 10
    - Values result in percentages from 0 to 100 with 0.1% resolution

CAN3, ID 0x0382

- Description: Battery Energy Status
- Length observed: 8 bytes
- Frequency on bus: Approximately 1 Hz
- Example frames: FC 9A EA 69 A9 00 A0 00
    - nominalFullPackEnergy: 76.4 kWh
    - nominalEnergyRemaining: 67.8 kWh
    - expectedEnergyRemaining: 67 kWh
    - idealEnergyRemaining: 67.7 kWh
    - energyBuffer: 4 kWh
    - energyToChargeComplete: 0 kWh
- Value descriptions:
    - nominalFullPackEnergy = (byte0 + ((byte1 & 0x03)<<8)) * 0.1
    - nominalEnergyRemaining = ((byte1>>2) + ((byte2 & 0x0F) * 64)) * 0.1
    - expectedEnergyRemaining = ((byte2>>4) + ((byte3 & 0x3F) * 16)) * 0.1
    - idealEnergyRemaining = ((byte3>>6) + ((byte4 & 0xFF) * 4)) * 0.1
    - energyToChargeComplete = (byte5 + ((byte6 & 0x03)<<8)) * 0.1
    - energyBuffer = ((byte6>>2) + ((byte7 & 0x03) * 64)) * 0.1
    - All formulas result in a value in kWh, resolution 0.1 kWh
    - energyBuffer appears to be the anti-brick buffer. This is NOT a below 0-miles remaining value.

CAN3, ID 0x0210

- Description: DC-DC Converter Status
- Length observed: 7 bytes
- Frequency on bus: Approximately 10 Hz
- Example frames: 00 00 D6 1B 1E 8A 00
    - inletTemperature: 19 C
    - inputPower: 432 W
    - outputCurrent: 30 A
    - outputPower: 414 W
    - outputVoltage: 13.8 V
- Value descriptions:
    - inputPower = byte3 * 16
        - Watts
    - outputCurrent = byte4
        - Amps
    - outputVoltage = byte5 / 10
        - Volts
    - inletTemperature = ((byte2 - (2 * (byte2 & 0x80))) * 0.5) + 40
        - degrees C
    - outputPower = byte4 * (byte5 / 10)
        - Watts

If you've found this data useful, and are going to utilize it in a project or elsewhere, please give credit. Was definitely a bit of work and very tedious to work out all of this info… and I'm just throwing it out on the web for all to see and use. Not asking for money or donations or anything.

I would definitely be interested in hearing about any projects that make use of this info!

Please drop by my thread on Tesla Motors Club where this all started, drop me a line via email, or PM there, or whatever works. Links to cool stuff below!

http://www.teslamotorsclub.com/showthread.php/58951-Let-the-hacking-begin-%28Model-S-parts-on-the-bench%29

http://wk057.solar/

https://www.youtube.com/user/wizkid057

wk057@skie.net

Again, this is not an official document, not affiliated with Tesla Motors, etc etc etc. Common sense should prevail when utilizing any of this info. It was derived from a limited data set of only a few vehicles and is probably not 100% accurate. There are certainly typos, errors, omissions, etc that could cause this info to be inaccurate, but I've done my best to check it as well as possible.

Most of this work was done between Tesla Model S firmwares v6.2 and v7.1.

There are a few other CAN IDs that other's have derived, like 0x06F2 (off the top of my head) for the individual battery cell group voltages. I purposely didn't include that in this initial document release since I didn't independently derive it prior to other's releasing it. I will, however, include it in the next release, with appropriate attributions. :) For now, here is a link to the thread on that:

http://www.teslamotorsclub.com/showthread.php/60509-Reading-Battery-Voltages-and-Temperatures-via-CAN-on-Model-S

Next up I'm going to work on decoding all of the thermal data available on the powertrain CAN. :)

-Jason / wk057

Original Link to this document and updates: http://skie.net/uploads/TeslaCAN/