READ • CHANGE • UNDERSTAND

OBD-II Learning Lab

From one ECU to a virtual vehicle

A complete reference with beginner lessons, diagrams, connector maps, signal scopes, model equations, worked cases, diagnostic examples and a staged route toward real firmware.

READ AND PREDICT	TRY AND EXPLAIN
Choose a chapter. Follow the diagram. Predict what should change before you move a control.	Disconnect a virtual sensor. Advance time. Read the computer’s response and try a repair.

OPEN THE MATCHING WORKBENCH

Extract the supplied ZIP. On Windows, double-click START_LAB.bat. On macOS or Linux, open a terminal in the OBDII_Lab folder and run the command below.

```
python3 run.py --serve
```

Then open http://127.0.0.1:8765 in a browser. Python 3.10 or newer is required; the core needs no extra packages. The PDF has a chapter tree for reading. The browser workbench runs the controls.

Version 0.3.0 · 9 November 2026

Generic teaching models, with separate Toyota architecture references. This is not a calibrated vehicle, a full emissions implementation or an original-firmware emulator.

Choose a chapter

A beginner’s book of vehicle signals 3

Work through one fault with the schematic workspace 20

Connectors, pin numbering and live probes 23

OBD-II and virtual vehicle learning lab 26

Worked cases: observations you can reproduce 39

OBD-II and ISO-TP: the diagnostic lesson 44

A path from this lab to a virtual vehicle and real firmware 51

Existing bases: what to reuse and where 55

EngDoc: a first interactive coolant-sensor worksheet 57

Sources and evidence boundaries 61

The bookmarks panel in your PDF viewer contains the document tree. In the live workbench, the tree stays beside the explanation, diagram and experiment controls. Start with the beginner chapters; open electrical details and message bytes when you want them.

The ZIP includes the simulator, this reading copy, offline HTML, editable sources, recorded experiments and verification notes. The complete guide also contains the model mathematics, connectors, diagnostic protocols, software reuse options and firmware roadmap.

A beginner’s book of vehicle signals

OBD-II Learning Lab · Version 0.3.0 · 9 November 2026

How to use this book

The interactive browser workbench combines a chapter tree, plain-language explanation, a diagram and a small experiment. Choose a chapter on the left, start its lesson, change one input and read what happened. Detailed electrical values and message bytes are optional deeper references. You can learn the main cause-and-effect ideas without typing them.

This PDF or printed copy contains the explanations and exercises. Its pictures and text are a reading reference. To move controls, disconnect a virtual sensor, advance time or collect a live trace, open the supplied browser workbench. A PDF viewer does not run the Python simulation.

The four questions to ask

1. What did I change?
2. What does the physical part actually do?
3. What information reaches the computer?
4. What does the computer decide, and why?

Controls in ordinary words

Control or display	Meaning
Start this experiment	Begin a fresh experiment with its defined starting conditions.
Watch it run / Pause and inspect	Let simulated time advance, or stop it so you can inspect the current state.
One small step / Run five seconds	Advance 0.1 seconds or five seconds of simulated time, then inspect the result.
Accelerator / Brake	Change the driver request in an experiment that permits manual inputs.
Disconnect the temperature sensor / Disconnect the front-left wheel sensor	Break the named virtual measurement path. The physical part remains in the model.
Reconnect / restore the input	Restore the currently injected measurement or communication fault.
Restart after repair	Restart the teaching controller after restoring its input; useful for the latched battery-message shutdown.
Stop the battery monitor reports	Remove or restore the repeated information from the battery computer. This is a communication failure, not a removed battery.
Read remembered engine warnings	Ask for the condition remembered by the engine computer and read its meaning.
Clear remembered engine warnings	Remove stored engine diagnostic history; it does not repair an active failure.
Advanced detail	Open the electrical values, signal identifiers, equations or protocol bytes when useful.

Use the currently available controls in the chosen lesson. A scripted lesson supplies some driver actions itself. Open Choose the starting conditions to change a starting temperature or charge level, then choose Start this experiment to apply those settings as a new comparison. The current fault model selects one injected fault at a time.

What is simulated?

The engine, battery, hybrid coordination, four longitudinal wheels and simplified brake controllers are working teaching models. Their values and thresholds are chosen for learning. The models have not been calibrated against a specific vehicle. The Toyota reference documents describe real historical architectures separately. Original firmware, a detailed charger, the full vehicle network and production stability-control behavior remain later stages.

1. Start here: change one thing

Aim: Follow a change from the driver or a sensor to the computer and then to the vehicle.

Think of this book as a workbench beside an explanation. The chapter tree stays on the left. On the right you can read, inspect a diagram, change one control and watch the result. You do not need to type computer commands or understand message codes to complete the lessons.

A sensor measures something. A vehicle computer checks the measurement and makes a decision. An actuator carries out a request, for example by changing engine effort or brake force. The engine, battery or wheels then respond. That response is measured again, so the loop repeats.

The drawing and the instruments describe the same running experiment. Start with the words: what changed, what the sensor reports, what the computer decided, and why. Ordinary measurements such as temperature and speed help you compare before and after. Detailed electrical values and message bytes are available when you want to look deeper.

This is a teaching vehicle. Its engine, battery and braking models are deliberately small enough to understand. Toyota documents provide separate examples of real vehicle architecture; the lab is not Toyota production software. The PDF is the reading and printing copy. Run the browser workbench to operate the controls.

Try it

1. Choose Start this experiment and leave the controls unchanged for a moment.
2. Choose Watch it run, then Pause and inspect. Notice that simulation time stops when paused.
3. Change one control, choose One small step and read the explanation of the result.
4. Choose Start this experiment again before comparing another change, so both experiments start from the same conditions.

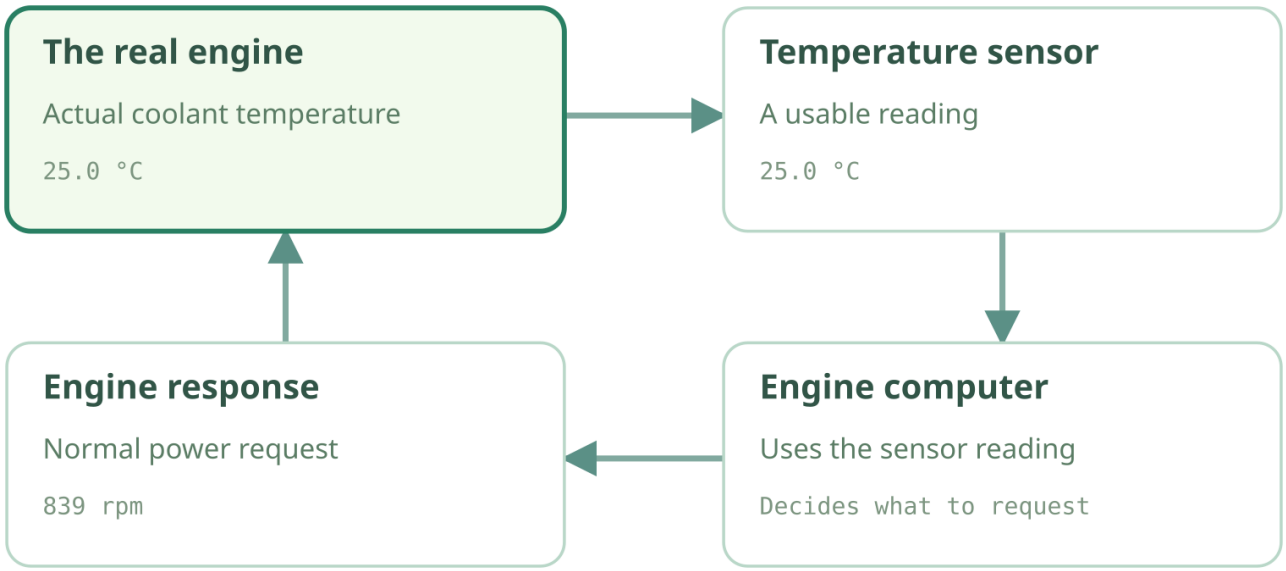
What to notice

- A command is a request, not an instant physical result.
- A measured value can be missing or wrong even when the physical part still exists.
- Faster playback advances simulated time faster on your screen; it does not change the model equations.

Explain it in your own words: When you change the pedal, what changes first and what takes longer?

2. An engine responds gradually

Aim: See why engine speed and temperature respond on different timescales.



Learning diagram: An engine responds gradually

The request, computer decision, physical engine and returning measurement form a repeated loop. This picture is a reference snapshot; the browser controls run the model.

The accelerator asks for more engine effort. The engine computer turns that request into a bounded command. The rotating parts have inertia: they resist a sudden change of speed. Engine speed therefore moves toward a new condition rather than jumping instantly.

The engine also produces heat. Coolant and metal store some of that heat, while cooling removes some. Temperature rises when heating is greater than cooling. It falls when cooling is greater. This is why a brief pedal change does not instantly heat or cool the engine.

This first experiment is an engine on a virtual test stand. There is no moving car attached to it. Engine speed can rise while vehicle speed stays at zero, and battery or wheel instruments are marked unavailable. That is an intentional smaller starting model.

Engine speed means how many turns the crankshaft makes in one minute. Temperature tells us how warm the coolant is. Fuel flow tells us how quickly fuel is being used. Read the explanation beside a measurement before interpreting a larger number as better or worse.

Try it

1. Choose Start this experiment for a fresh engine lesson.
2. Choose Watch it run with a modest accelerator setting and watch engine speed settle.
3. Increase the accelerator, then compare the immediate speed change with the slower temperature change.
4. Pause and inspect the engine loop: request, computer decision, physical response and measurement.

What to notice

- A turning engine has stored motion; it does not stop or accelerate without a physical cause.
- Coolant warms gradually because the model includes heat storage.
- The engine stand intentionally has no installed battery controller or wheel controller.

Explain it in your own words: Why can engine speed change quickly while the coolant temperature changes slowly?

3. How the computer knows the temperature

Aim: Separate the real temperature, the measurement and the value used for control.

The computer cannot feel the coolant directly. A temperature-sensitive component changes its electrical resistance as it warms. In this circuit, warming reduces resistance. Together with a fixed resistor inside the computer, it makes a voltage that the computer can measure.

The computer turns the measured voltage into a temperature estimate. In normal operation that estimate stays close to the actual coolant temperature. Small differences are expected because this model rounds an electrical measurement to a finite set of digital steps.

Keep three ideas separate. Actual temperature is what the physical model says the coolant is doing. Reported temperature is what the sensor path lets the computer estimate. Control temperature is the value the computer chooses to use. They agree closely when healthy but can differ after a fault.

The connector drawing shows a signal path and a return path. Opening the signal path breaks the measurement circuit; it does not remove heat from the coolant. The labels X1 and J-ECT are names chosen for this teaching harness, not the connector numbers of a particular Toyota.

Try it

1. Start with the sensor connected.
2. Change the starting temperature when the lesson offers that setting, then reset or apply the new starting condition.
3. Compare the actual temperature with the reported temperature and the temperature used by control.
4. Select the wire or sensor in the drawing and read its explanation before opening the advanced electrical details.

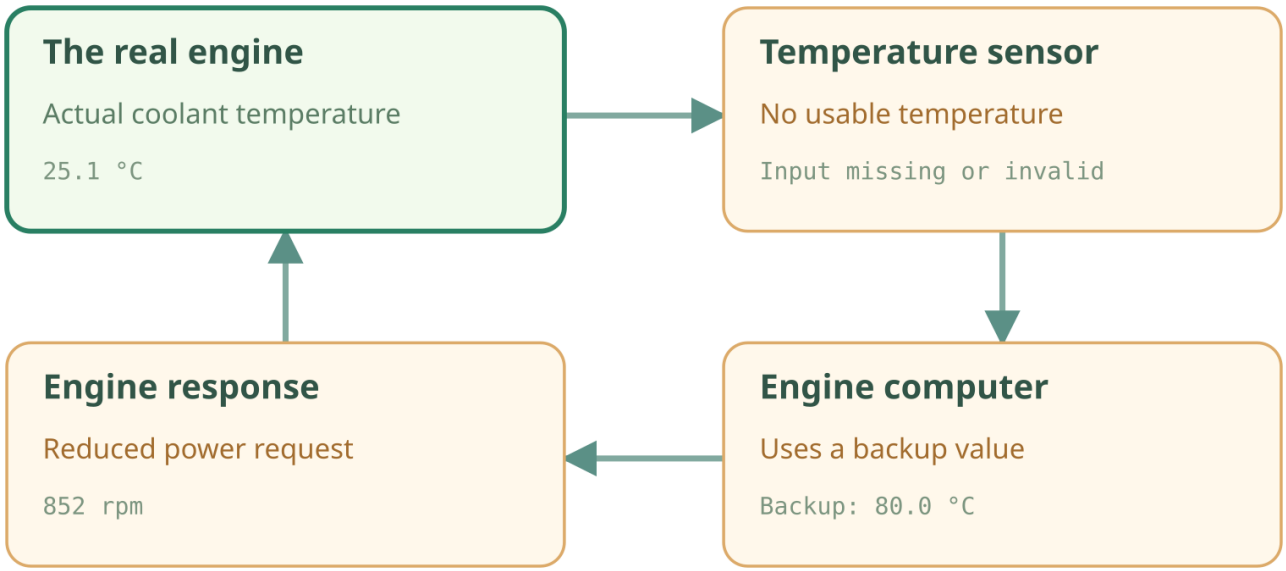
What to notice

- A healthy sensor report should follow the physical temperature.
- For this particular temperature-sensor circuit, warmer coolant produces a lower signal voltage.
- The actual wire voltage and the rounded computer representation are separate observations of the same circuit.

Explain it in your own words: If a sensor reading changes, must the real temperature have changed?

4. What happens when a sensor is disconnected?

Aim: Explain a sensor fault without mistaking it for a sudden physical temperature change.



Learning diagram: What happens when a sensor is disconnected?

A broken measurement path changes what the computer knows, while the physical coolant remains in the model. This is a reference snapshot; disconnect the sensor in the browser to observe the timed response.

Begin with a healthy engine, then disconnect the coolant sensor using the lesson fault control. The sensor branch no longer carries current. The computer input is pulled high by its internal resistor. That high input cannot be treated as a believable temperature measurement.

The real coolant remains warm. The model shows an invalid cold reading in the sensor report, but its plain-language note explains that this is a failed measurement. A value marked invalid is not evidence that the physical engine reached that temperature.

The engine computer waits briefly to see whether the bad input persists. In this lesson, after one fifth of a second it uses a substitute temperature and reduces its requested engine effort. After half a second it stores a confirmed warning and turns on the engine warning lamp. These short waiting times are teaching choices, not a manufacturer calibration.

Reconnecting the sensor repairs the signal path. The current measurement can recover while the stored warning remains as a record of what happened. Clearing warning memory removes that record. Clearing memory while the sensor is still disconnected leaves the cause in place, so the warning can return.

Try it

1. Choose Start this experiment, then Watch it run until you see a normal measurement.
2. Choose Disconnect the temperature sensor, then One small step.
3. Compare actual temperature, invalid sensor report and substitute control value as time advances.
4. Read the stored warning in words. Repair the sensor connection, then clear the engine warning memory.
5. For a second experiment, clear the warning while still disconnected and observe why it returns.

What to notice

- The failure begins in the measurement path, not in the physical coolant.
- The substitute is a chosen control value; it is not a new measurement.
- Repairing a cause and clearing its history are separate actions.

Explain it in your own words: Why should clearing a warning fail to repair a disconnected wire?

5. The battery computer sets limits

Aim: Understand why the vehicle may refuse a request even when the driver keeps pressing the pedal.

The battery stores energy, but it cannot provide or accept unlimited power. Its controller watches charge level, temperature and voltage-related inputs. It tells the rest of the vehicle how much battery power may be used for driving and how much may be returned during braking.

Charge level is an estimate of how full the battery is. Low charge can make the hybrid controller request more engine use. A nearly full battery may have little room to accept recovered braking energy, so the friction brakes must do more of the stopping work.

The battery computer sends repeated status reports to the hybrid controller. A report needs to be both believable and recent. If updates stop, the last received numbers may still look reasonable. The workbench marks them as old while the actual battery model can continue changing.

Try stopping the battery status messages. In this model, a report older than roughly three tenths of a second causes the hybrid controller to inhibit driving effort and request disconnection of the high-voltage

path. That stop is latched: restoring messages alone does not remove it. Restore the messages, then choose Restart after repair or start a fresh experiment. This is the selected teaching policy, not a promise about every real vehicle.

Try it

1. Choose Start this experiment, then Watch it run, and observe when the engine is requested in this low-charge lesson.
2. Compare the current physical battery state with the latest report received by the hybrid controller.
3. Use the fault control to stop battery status updates, then advance the simulation.
4. Watch the report become old and read why the controller stops allowing driving effort.
5. Choose Reconnect / restore the input, then Restart after repair to clear the teaching shutdown latch.

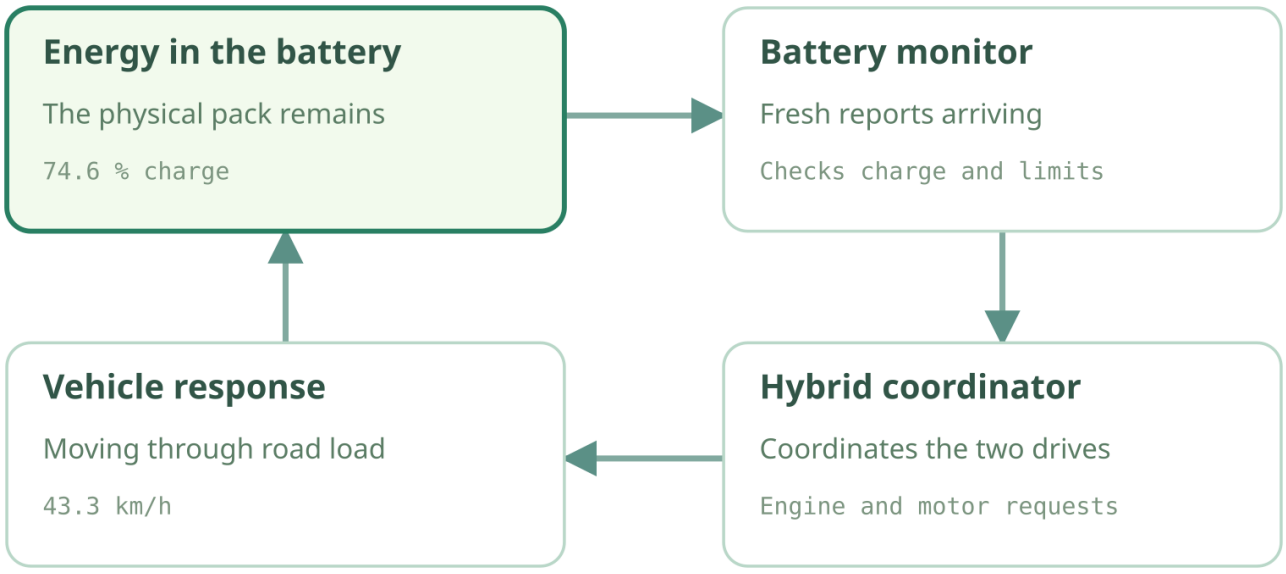
What to notice

- A battery limit is permission, not a demand to use that much power.
- A held old value is different from a live measurement and different from zero.
- A stored shutdown condition can remain after the immediate communication problem is repaired.

Explain it in your own words: Can an old, believable battery report be trusted as a current report?

6. The engine and motor work together

Aim: Explain how driver demand and battery limits change the use of engine, motor and brakes.



Learning diagram: The engine and motor work together

The hybrid coordinator shares a driver request between the engine and motor within battery limits. This is a reference snapshot; the browser experiment changes the values as it runs.

The hybrid supervisor coordinates the engine, electric motor and battery. It receives the driver request and checks what the other components can provide. It can request electric driving, engine assistance, or a mode that tries to preserve the battery charge.

During acceleration, stored battery energy and fuel energy can become motion. During braking, the motor can act as a generator and return some motion energy to the battery. This is regenerative braking. Some energy is still lost, and the battery has limits on how much power it can accept.

Friction brakes turn motion energy mainly into heat. They provide the part of the requested braking that regeneration cannot supply. A fuller battery can therefore mean less recovered energy and more friction-brake work during the same braking request.

The diagram separates requests, information and energy flow. A line carrying a power value does not necessarily represent a physical harness wire. This lab uses a generic parallel-hybrid arrangement. A Toyota Prius power-split arrangement is explained separately in the reference material.

Try it

1. Choose Start this experiment, then Watch it run, and read the plain-language hybrid operating mode.
2. Choose Pause and inspect during acceleration and identify the engine and motor contributions.
3. Continue into braking and observe whether the battery is supplying energy or receiving it.
4. Compare with the nearly-full-battery braking lesson in the experiment list.

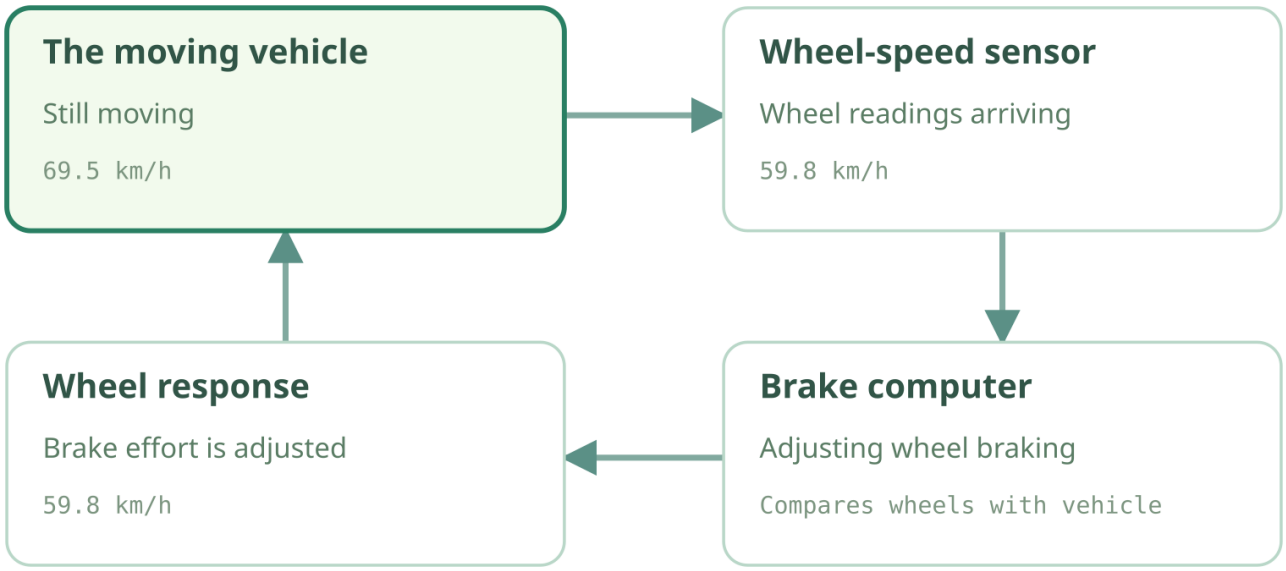
What to notice

- The same pedal request can lead to a different decision when battery permission changes.
- Returned battery energy is smaller than the mechanical energy recovered because the model includes conversion losses.
- The motor and friction brakes must share a physically meaningful braking task.

Explain it in your own words: When the battery cannot accept more charging, where must the braking energy go?

7. Why ABS watches the wheels

Aim: Compare vehicle motion with wheel motion and see the brake controller adjust its request.



Learning diagram: Why ABS watches the wheels

The brake controller compares wheel measurements with the vehicle motion and adjusts brake effort. This is a reference snapshot; use the browser for the moving wheel and braking experiment.

A moving vehicle and a rotating wheel are related, but they are not the same thing. During hard braking a wheel can slow too much compared with the vehicle. It may slide across the road instead of rolling normally. The workbench calls this difference braking slip.

The anti-lock brake controller watches the wheel-speed inputs and adjusts a brake multiplier. In this model it reduces braking on a wheel when slip becomes too large, then allows braking to build again. The multiplier is a relative command, not measured hydraulic pressure.

The lesson begins with less grip on one side of the road than the other. The wheels therefore do not all respond alike. Compare each wheel with the vehicle body speed and watch which brake commands change.

The lab also has a simplified yaw-assistance model. Yaw means the vehicle turning left or right around a vertical axis. The model demonstrates a feedback idea, but it does not reproduce a full vehicle stability-control system or all road conditions. Compare the supplied case carefully; it does not prove that ABS always makes every stopping distance shorter.

Try it

1. Choose Start this experiment, then Watch it run, for the unequal-grip braking lesson.
2. Choose Pause and inspect after braking begins and compare vehicle speed with each wheel speed.
3. Inspect a wheel-speed signal and the corresponding brake multiplier.
4. Choose Start this experiment for a fresh comparison, then use the next chapter to see what changes when a wheel signal is missing.

What to notice

- A wheel turning much too slowly can be sliding while the vehicle is still moving.
- The controller changes individual brake commands because the wheels can have different grip.

- A comparison is meaningful only when starting speed, road conditions and driver request are held consistent.

Explain it in your own words: Why does the controller need wheel speed as well as the driver brake request?

8. Turn off one wheel-speed signal

Aim: Distinguish a sensor failure from a physical wheel locking.

Disconnect the front-left wheel-speed signal in the virtual lesson. The physical wheel remains in the vehicle model and may still be rotating. Its measured input becomes unavailable. The diagram and notes keep those two facts separate.

The selected teaching policy disables both anti-lock modulation and the simplified yaw-brake assistance when that wheel input is missing. The driver brake request still produces friction braking. Losing assistance does not mean the model has removed every brake.

A blank or missing sensor value must not be interpreted as a measured zero. Zero would mean the sensor successfully reported a stopped wheel. Missing means the controller does not have a usable report from that input.

Reconnect the sensor and observe the measurement return. Assistance can resume when its speed and control conditions are met, while the module fault record remains part of the experiment history. Choose Start this experiment when you want to begin a new clean comparison.

Try it

1. Start the wheel-sensor lesson with the vehicle already moving.
2. Apply or keep a brake request and observe the healthy signals first.
3. Choose Disconnect the front-left wheel sensor, then One small step.
4. Compare the physical wheel speed, missing sensor report and changed brake-controller status.
5. Choose Reconnect / restore the input, then Start this experiment before measuring a fresh before-and-after comparison.

What to notice

- The wheel can keep rotating after its sensor signal disappears.
- The missing input is shown as unavailable, not as a fabricated zero.
- In this model friction braking continues while the selected electronic assistance is disabled.

Explain it in your own words: What would go wrong if a missing sensor were silently displayed as a zero-speed wheel?

9. Ask the computer what happened

Aim: Read current measurements and stored warnings in ordinary words.

A diagnostic tool asks a vehicle computer questions. Useful beginner questions are: What temperature do you currently report? Are you using a substitute value? Have you stored a warning? The workbench explains the answer before showing the underlying message representation.

A stored trouble code names a condition the computer detected. It is evidence to investigate, not proof that one particular physical part must be replaced. A disconnected sensor, a broken wire or another input-circuit problem can be related to the same kind of detected condition.

Current measurement, active failure and stored history are different. A repaired signal can give a normal reading while the computer still remembers the earlier fault. Clearing engine warning memory removes that record; it does not repair a connection or guarantee that every monitor has completed.

Generic OBD-II mainly concerns emissions-related reporting. In this lab the engine scan questions do not automatically reveal every battery or brake-controller fault. Read those module explanations in their own lessons. The optional advanced view shows message bytes for people who want to learn the protocol later.

Try it

1. Choose Start this experiment, then Watch it run, and read the explanation as the coolant-sensor fault develops.
2. Ask for the current coolant reading and compare it with the physical temperature.
3. Choose Read remembered engine warnings and compare the pending and stored history.
4. Repair the sensor path, choose Clear remembered engine warnings, and check the current state again.

What to notice

- An invalid reading needs its validity note, even if it is displayed as a number.
- Stored history can outlast the immediate fault.
- Engine emissions diagnostics and other vehicle-module reports have different scopes.

Explain it in your own words: Does a stored warning identify the exact damaged part, or the problem the computer detected?

10. From a small lesson to a virtual vehicle

Aim: See how the workbench can grow while keeping each change understandable.

A useful next step has one clear question. For example: Can a new battery model explain a measured current trace? Can a different engine controller react to the same temperature-sensor failure? Keep the surrounding experiment steady so the effect of the changed part is visible.

For each new subsystem, write down what it measures, what it controls, the units, how often it updates and what happens when information is missing. Add a clear diagram and one reproducible healthy case before adding several interacting faults.

The current workbench can exchange high-level driver and plant information through its supplied software interfaces. Original vehicle firmware is a later task. Such a program expects a particular processor, memory layout and hardware peripherals; a diagram or an arbitrary firmware file is not enough to run it correctly.

The full theory and reference chapters remain available in the document tree, including equations, diagnostic bytes, existing software bases, Toyota examples and the firmware roadmap. The supplied

EngDoc format has now been inspected. The separate native coolant worksheet lets you change a temperature and disconnect its measurement path; docs/ENGDOC_INTEGRATION.md explains its scope. It calculates a static circuit and does not replace the timed browser model or run vehicle firmware.

Try it

1. Choose one component and one question to investigate.
2. Record its inputs, outputs, units, update period and invalid-input behavior.
3. Run a healthy baseline, change one assumption and compare the same measurements.
4. Use the theory and reference chapters when you need the equations or implementation detail behind a result.

What to notice

- A more detailed model should explain an additional behavior, not merely add more blocks.
- A claimed real-vehicle parameter needs a source and vehicle-version context.
- The PDF can explain a lesson; the browser workbench is where its interactive controls run.

Explain it in your own words: What measurements and timing would a new controller need before it could replace this teaching block?

Three guided fault exercises

Exercise A: disconnect the coolant sensor

Prediction: The real coolant should remain near its previous temperature, but the engine computer should lose a believable temperature measurement.

1. Start a healthy engine lesson and observe the three temperatures: physical, reported and used for control.
2. Choose Disconnect the temperature sensor. Advance time with One small step.
3. Read the invalid-measurement explanation. Continue until the substitute control value and stored warning appear.
4. Choose Reconnect / restore the input. Check that the current reading recovers even if a warning remains stored.
5. Clear engine warning memory. Repeat once without repairing the connection and explain why the warning returns.

Expected result in this model: An open sensor branch carries no current. Its input voltage reaches the reference level. The displayed sensor-temperature sentinel is invalid, while the physical coolant stays warm. The chosen fallback begins after 0.20 seconds of persistent high input; confirmation follows after 0.50 seconds. These timers and the substitute temperature are lesson settings. The detailed reference identifies the engine trouble code as P0118; learning the code is optional.

Exercise B: turn off a wheel-speed input

Prediction: Losing the report should not physically remove or stop the wheel.

1. Start the moving-wheel lesson and observe the actual front-left wheel speed and its measured report.
2. Apply a brake request and disconnect the front-left sensor signal.
3. Compare the still-existing wheel state with the unavailable sensor input.
4. Read the brake-controller decision. Friction braking remains, but this teaching policy disables anti-lock modulation and its simplified yaw assistance.
5. Choose Reconnect / restore the input, then Start this experiment before measuring a new clean stopping comparison.

Expected result in this model: The missing measured speed is blank or unavailable, never a fabricated measured zero. Assistance can resume after reconnection when its normal speed/control conditions are met. The stored module record remains part of experiment history until reset.

Exercise C: stop battery status reports

Prediction: The last reported value can look normal while it becomes too old to trust.

1. Start a hybrid lesson and allow the vehicle to become ready.
2. Compare the current physical battery value with the value received by the hybrid controller.
3. Stop battery reports. Watch report age grow while the last received values remain held.
4. Continue until the hybrid controller declares the reports too old and inhibits driving effort.
5. Choose Reconnect / restore the input, then Restart after repair to remove the selected latched shutdown.

Expected result in this model: Reports older than about 0.30 seconds trigger the chosen communication-failure policy at the next supervising update. Restored communication alone does not unlatch that shutdown. This exercise removes a stream of information; it does not disconnect a real high-voltage connector or simulate a physically absent battery.

Read the results without being misled

What you see	What it means
Current and usable	The model currently supplies a usable value for this signal.
Invalid measurement	A value may still be displayed to explain the fault, but it is not a trustworthy measurement of the physical quantity.
Substitute used	The computer is using a documented replacement value after a detected problem.
Old report	The last received value is held; no recent update has arrived.
Unavailable	There is no usable value, or the chosen lesson does not contain that subsystem.
Stored warning after repair	The current input may be healthy while diagnostic history still records the earlier fault.

The signal scope records samples every tenth of a second. Some controller decisions happen between those saved samples. Read the event explanation and current state when checking exactly when a fault was noticed. Curves help compare changes; they are not a recording of every wire voltage transition.

A small amount of mathematics, in words

Engine motion: More turning effort than resistance makes engine speed rise. Greater rotational inertia makes the same extra effort change speed more slowly.

Temperature: Heating adds stored thermal energy. Cooling removes it. The temperature changes according to the difference and the amount of heat the modeled engine/coolant can store.

Battery charge: Current flowing out reduces the stored charge; current flowing in increases it. The model also accounts for internal electrical loss. Charge percentage is not a complete battery-health assessment.

Braking: The brakes must remove motion energy. Regeneration can return part of it to the battery within permission and efficiency limits; friction brakes supply the remainder.

Measurement: A physical quantity becomes an electrical input, then a digital representation, then an interpreted value. A fault or a rounding step can affect one layer without changing the others.

The full Learning Guide gives equations, units and worked calculations for these relationships. The Working Cases reports results generated by the supplied model.

Glossary: ordinary words first

Term	Plain meaning	More context
Sensor	A part that measures something.	Examples are coolant temperature and wheel speed. A sensor report may be valid, missing or unreliable; the physical part being measured can still exist.
Engine computer / ECU	The computer that checks inputs and requests engine actions.	ECU means electronic control unit. In this lab, the engine ECU is a simplified software controller with a temperature-fault monitor.
Actuator	A part that carries out a control request.	Examples include a brake mechanism or a motor. The requested action and the actual physical result can differ because of limits and delays.
Physical model / plant	The simulated engine, battery, wheels and their physical behavior.	The plant changes according to equations. It is distinct from the controller that makes decisions.
Signal	A value or state passed from one part of the model to another.	A signal can mean a measurement, a request, a permission or a status. A line on a logical diagram is not automatically a physical wire.
Feedback	Using the measured result to decide what to do next.	The controller requests an action, the physical model responds, and a sensor reports the response. The loop repeats.
Coolant	The fluid represented by the engine temperature model.	The lab uses one average temperature. It does not model each coolant passage, thermostat or local hot spot.
Thermistor / NTC	A resistor whose resistance changes with temperature.	For this NTC model, warming lowers resistance. The computer measures a voltage produced by that changing resistance.
Voltage	An electrical difference measured between two points.	The lesson sensor signal is measured relative to the sensor return. A high voltage can mean a broken sensor path in this particular circuit.
Resistance	How strongly a part opposes electrical current.	A connected thermistor has a finite resistance. An open branch is shown as missing or invalid rather than a made-up very large reading.
Current	The rate of electrical charge flow.	In the battery model, current flowing out supplies energy for the vehicle; current flowing in charges the battery. The display explains the direction.
ADC	The computer input that turns a measured voltage into digital steps.	ADC means analog-to-digital converter. Finite steps cause small rounding differences even in a healthy ideal measurement.
Valid / invalid	Whether a measurement is usable for its stated purpose.	Invalid does not mean the number must disappear. It means the learner and controller must not treat it as a reliable measurement.
Unavailable	No usable value is present for this signal here.	A module may not be installed in the chosen lesson, or a sensor report may be missing. Unavailable is different from a measured zero.

Term	Plain meaning	More context
Fallback / substituted value	A value the controller chooses when the normal measurement cannot be used.	A substitute supports the selected failure response. It is not a fresh sensor measurement and does not change the actual physical temperature.
Stale / old report	Information that has not been updated recently enough.	A last received battery report can remain believable while it is too old to use for current power permission.
Warning / trouble code	A stored name for a condition the computer detected.	It helps identify a problem area. It does not by itself prove which physical part caused the condition.
Battery computer / BMU	The controller that reports battery condition and allowed power.	BMU means battery management unit. BMS is another common battery-management name. This lab uses a simplified pack model.
Charge level / SOC	How full the battery is estimated to be.	SOC means state of charge. It is commonly shown as a percentage; it is not the same as the battery voltage or total lifetime health.
Power	How quickly energy is being transferred.	A power limit is a permitted rate, not a request to use all of it. Battery power, mechanical motor power and recovered braking power can differ.
Contactor	A controlled switch in the high-voltage power path.	The lab represents its state and a simplified precharge sequence. It does not simulate relay arcs or provide physical wiring instructions.
Hybrid supervisor	The computer coordinating engine, motor, battery and braking requests.	Its decision depends on driver demand and component permissions. This lab demonstrates a generic strategy rather than Toyota software.
Regenerative braking	Slowing the vehicle while returning some motion energy to the battery.	The motor acts as a generator. Battery acceptance and conversion losses limit how much energy is recovered.
Friction braking	Slowing the vehicle by turning motion energy mainly into heat at the brakes.	It supplies braking that regeneration cannot provide. In the missing wheel-sensor lesson it remains present when electronic assistance is disabled.
ABS	Anti-lock braking assistance.	ABS compares wheel behavior with vehicle motion and changes individual brake commands to reduce excessive braking slip in the teaching model.
Slip	The difference between wheel rolling speed and vehicle speed during braking.	A wheel turning too slowly for the vehicle speed may be sliding. The model uses a regularized ratio to express that difference.
Yaw / ESC / ESP	Vehicle turning motion and the assistance intended to influence it.	Yaw is rotation left or right around a vertical axis. ESC and ESP are stability-control names; this lab includes only a simplified yaw-assistance demonstration.
OBD-II	A standard way to inspect emissions-related vehicle information.	The lab includes current engine readings and engine warning memory. Other module information may require separate manufacturer-specific diagnostics in a real vehicle.

Term	Plain meaning	More context
CAN	A communication method used by vehicle computers.	The lab can display diagnostic frames and exchange them in a virtual-bus example. Physical voltage waveforms, wire faults and real bus arbitration are not modeled.
Firmware	The program intended to run inside a particular electronic controller.	It depends on its processor and hardware environment. Running original firmware is a future integration task, not a current feature of this workbench.

References and evidence boundaries

The executable code and tests describe what this release actually does. The sources below support general principles and terminology; they do not validate the lab’s synthetic parameters or chosen failure policies. The complete source and Toyota-document inventory remains in Sources.

- California Air Resources Board: On-Board Diagnostic II Systems Fact Sheet. Supports the emissions-related scope of OBD-II.
- Texas Instruments: Monitoring NTC Thermistor Circuit With Single-Ended ADC, SBAA338A. Supports the temperature-resistance-divider and ADC principles. The lesson values are selected separately.
- Toyota Motor Corporation: Toyota Develops New Hybrid System, 17 April 2003. A historical manufacturer example of hybrid operation and power splitting, separate from this generic plant.
- Bosch Mobility: Electronic stability program. Supports the role of wheel-speed, steering and yaw information in stability assistance. The lab’s reduced model is not Bosch software.
- QEMU: Arm System emulator, and Renode: Peripheral modeling guide. Explain why firmware needs a suitable processor and peripheral environment.

Continue with Schematic Workspace, Connector Reference, Protocol, Existing Bases, and Firmware Roadmap when you are ready for more detail.

Work through one fault with the schematic workspace

Version 0.3.0 · 9 November 2026

In the Learning book, choose **Instruments** or **Open detailed instruments** to use these five screens. The learning book and instruments share the same experiment state.

The task workspace follows one complete causal path: **coolant temperature → thermistor resistance → input voltage → ADC measurement → ECU decision → OBD response**. Its diagrams and instruments share the running simulation state. This makes it possible to probe a signal, interrupt the sensor path, advance time and see why a diagnostic code appears.

The interface borrows the useful inspection pattern of a block-diagram environment: a system view, selected-object inspector, connector view and signal scope. This version has fixed, documented model connections. It does not provide arbitrary block placement, wire editing, code generation or a general numerical solver editor.

Five screens, one running experiment

Screen	What to use it for
01 · Task workspace	Follow the six-step ECT exercise; inspect a system or circuit schematic, probe signals, compare traces and read controller events
02 · Connectors	Select a physical DLC reference or illustrative J-ECT/X1 view; inspect pin orientation, function and live sensor values
03 · Signal list	Search 69 cataloged signals; inspect current value, units, validity and nominal update period; add one of 67 numerical/digital channels to the scope
04 · Scan tool	Send an OBD application request and inspect the generated CAN/ISO-TP transcript and decoded result
05 · Dashboard	View vehicle, engine, battery and braking behavior across the existing lessons; export the complete run

The schematic selector includes an engine sensor/ECU loop, the ECT electrical circuit, hybrid control and power, and ABS/braking signals. These are views of the model’s boundaries. A connection representing a controller request is not automatically a physical wire or a verified OEM CAN message. The connector reference documents the physical-versus-illustrative distinction separately.

The six-step ECT task

1. **Healthy baseline.** Start the guided task and select **ECT electrical circuit**. The task runs for 1 s from a selected 25°C coolant initial condition. A powered 5 V reference feeds a 2,200 Ω pull-up and the coolant thermistor returns to an ideal sensor ground. Near 25°C, resistance is about 2,500 Ω, physical node voltage is about 2.6596 V and ADC count is near 2178; small thermal changes during the baseline can move these values. These are model outputs, not measurements from a Toyota.

2. **Open circuit.** Disconnect the modeled sensor path. The ECU/harness side remains attached to the pull-up. Its physical voltage rises to 5.000 V, current becomes zero and ADC count reaches 4095. Actual coolant temperature stays close to its previous value. Probe `coolant_c`, `sensors.ect_c` and `sensors.ect_used_c` to distinguish the plant, interpreted measurement and controller input.
3. **+0.20 s.** Advance the fault timer to the pending/fallback threshold. The teaching ECU marks P0118 pending, substitutes 80°C for control and applies a 0.65 torque multiplier. Its sensor report remains the invalid −40°C sentinel. The monitor checks an ADC input at or above 99% of the reference, or 4.95 V with the default 5 V source.
4. **+0.30 s.** Advance to 0.50 s of persistent fault. P0118 becomes confirmed and the MIL turns on. These short timers support observation in a lesson; they are not a manufacturer calibration or regulatory trip-confirmation rule.
5. **Read DTC.** Inspect service `03` for the confirmed code. Compare `07` for pending codes and `01 05` for coolant data. In this model, the open sensor yields application reply `41 05 00`; the PID conversion `A − 40` gives −40°C. The event timeline explains when the controller changed state.
6. **Repair & clear.** Reconnect the sensor path and clear diagnostic memory with service `04`. Check that the input returns to a plausible voltage, the fallback ends and fault memory clears. Clearing alone while the path is still open does not repair the cause: advance time and the monitor will set the fault again.

The task controls create a controlled demonstration. The existing scripted **ect_open** lesson instead introduces the open at simulation time 3.00 s. Use that lesson for a reproducible unattended recording; use the task buttons to inspect each transition interactively.

Read the instruments carefully

The scope plots saved samples at **100 ms** intervals. Engine sensor/monitor updates occur every **10 ms**, while the plant advances in **2 ms** steps. UI playback speed changes wall-clock presentation, not those model periods. A transition may occur between plotted samples; the controller event timeline and current monitor timer are better evidence of its exact simulated timing.

Each scope row has its own signal name and units, with up to six channels at once. Adding volts, ADC counts and degrees Celsius does not put them on one common physical scale. Click a trace to inspect a saved sample, choose a time window, and export selected traces as CSV. The signal catalog distinguishes valid, invalid, substituted, stale and unavailable values. Invalid, substituted and stale finite samples use dashed scope segments. An open-path resistance is `null`, marked invalid, because no finite measurement is reported; null samples leave gaps instead of drawing a fabricated resistance.

Physical node voltage and ADC-reconstructed voltage should usually be close. At 25°C they are 2.659574 V and 2.659341 V respectively; the latter comes from count 2178. Their difference is expected quantization. The connector reference shows the hand calculation and identifies which probe belongs to which point.

The diagnostic console models application data and in-memory classical CAN/ISO-TP frames. It does not simulate CAN bit timing or electrical waveforms. Consequently, clicking DLC CAN-H or CAN-L provides a protocol explanation rather than a made-up analog trace.

How this supports later expansion

Keep each future subsystem explicit about its physical state, sensors, controller inputs, outputs, update period and validity rules. A BMU lesson can follow current integration into SOC and power limits; a braking lesson can follow wheel-speed validity into slip control and pressure requests. More detailed models can then replace a block without changing what a student probes at its boundary.

Real firmware remains a future integration path. The current inspector is not a firmware editor or an ECU flashing tool. Executing firmware will require an appropriate processor/peripheral environment, input/output mapping and an agreed simulation clock. See Firmware roadmap for that progression and Protocol for the implemented diagnostic subset.

The numerical claims in this guide come from this release's executable teaching model, primarily `obdii_lab/core.py` and `obdii_lab/instrumentation.py`. The physical connector claims are sourced in the connector reference; they do not validate the synthetic sensor parameters or lesson pin assignments.

Connectors, pin numbering and live probes

Version 0.3.0 · 9 November 2026

The **Connectors** screen separates a referenced vehicle diagnostic connector from the invented connector assignments used by this lesson. A connector drawing is useful only when its viewing direction and applicability are clear. Every view therefore identifies its orientation, function and evidence status.

1. Vehicle diagnostic connector

The default drawing shows the **mating face of a J1962 Type A vehicle socket**, looking into the socket with its wide edge at the top. Pin 1 is at the upper left, pin 8 at the upper right, pin 9 at the lower left and pin 16 at the lower right. Elm Electronics illustrates this vehicle-connector numbering in Figure 8 of its ELM327 data sheet. [C1]

Show rear / wire side mirrors the drawing left to right to show the opposite side of the *same connector*. A view of a mating cable plug is a different reference. Before using an actual connector, check its own drawing and whether the manufacturer labels the mating face or wire side. The lab's rear view is an orientation aid, not a dimensioned harness assembly drawing.

Pin	Reference function	Treatment in this lab
4	Chassis ground	Role identified; electrical ground offsets not solved
5	Signal ground	Role identified; not the separately named ECT sensor-return cavity
6	CAN High for ISO 15765-4	Diagnostic frames; analog voltage not modeled
14	CAN Low for ISO 15765-4	Diagnostic frames; analog voltage not modeled
16	Vehicle battery positive	Supply voltage not simulated
2	J1850 Bus +	Legacy reference; inactive
10	J1850 PWM Bus −	Legacy reference; inactive
7	ISO K-line	Legacy reference; inactive
15	ISO L-line where applicable	Legacy reference; inactive
1, 3, 8, 9, 11, 12, 13	No vehicle-specific role assigned here	No simulated signal

The manufacturer pin guide supports these reference functions and distinguishes the protocols. Pins 6 and 14 belong to the selected CAN profile; their use must not be generalized to every OBD-II protocol. [C2] Figure 9 of the Elm data sheet provides a corresponding adapter circuit. [C1]

Selecting a CAN pin explains its frame-layer boundary. There is no numerical pin-voltage probe, dominant/recessive waveform, termination measurement or bus arbitration model. Pin 16 does not display an invented 12 V measurement, and the hybrid traction-pack voltage is not assigned to it.

2. Lesson sensor and ECU assignments

J-ECT and X1 are fictional teaching labels. They do not identify Toyota connector part numbers or establish a cavity assignment for any real ECU. “THW” is used only as an ECT signal label in this circuit. Applying the lesson to an actual vehicle requires the applicable electrical wiring diagram and component documentation.

Lesson point	Meaning	Live signal
J-ECT.1	Harness-side ECT input, joined to X1.2	`sensors.ect_node_v`
J-ECT.2	Sensor return, joined to X1.3	`sensors.sensor_return_v`
X1.1	Internal reference test point	`sensors.ect_reference_v`
X1.2	Physical ECU input node	`sensors.ect_node_v`
X1.3	Ideal sensor-return reference	`sensors.sensor_return_v`

J-ECT shows the cavities of the harness-side connector. The open-path switch lies after J-ECT.1 toward the thermistor; the harness remains connected to the ECU pull-up. An isolated sensor-side terminal is not separately modeled. X1.1 is an internal reference test point, not an assertion that a real ECU exposes a 5 V supply at cavity 1.

Select a pin with the mouse, or focus it and press Enter or Space. **Add this signal to scope** adds its channel to the task workspace. The ECT input also offers **Probe ADC voltage instead**, because the physical voltage and its digital reconstruction are different quantities. The mapping table provides an equivalent keyboard-accessible route. **Export connector SVG** saves the displayed orientation and selection.

3. A numerical circuit example

The default teaching model uses a 5 V reference, a 2,200 Ω pull-up and a beta-model NTC with 2,500 Ω at 25°C and beta 3,500 K. The return is an ideal 0 V reference. These are selected model parameters, not Toyota calibration values. See `obdii_lab/core.py`, `Parameters` and `ect_circuit`, for the executable definition. With absolute temperature `T` in kelvin, resistance follows:

$$R(T) = 2500\exp\left[3500\left(\frac{1}{T} - \frac{1}{298.15}\right)\right]\ \Omega.$$

At 25°C, the thermistor is 2,500 Ω. The divider gives

$$V_{node} = 5\frac{2500}{2200 + 2500} = 2.659574\ \text{V}.$$

The branch current is 1.06383 mA. A 12-bit ADC gives `round(2.659574 / 5 × 4095) = 2178`, reconstructed as **2.659341 V**. That small difference is quantization, not a voltage drop along the wire. J-ECT.1 and X1.2 therefore show the same physical value, while the ADC channel shows the reconstruction.

For a valid reconstructed voltage `Vm`, recover resistance as `Rm = 2200 × Vm / (5 − Vm)` and temperature as `1 / [1/298.15 + ln(Rm/2500)/3500] − 273.15` °C. Validity checks precede this inverse calculation; the open circuit is not interpreted by dividing by zero.

Opening the path makes current zero and pulls the harness node to **5.000 V**, giving ADC count **4095**. Version 0.2 replaces the earlier 4.98 V placeholder with this ideal open-circuit result. The temperature channel becomes invalid; the displayed −40°C is a diagnostic sentinel, not evidence that the coolant suddenly froze. The controller's later 80°C fallback is a third, explicitly separate value. The workspace guide follows this transition through the diagnostic monitor.

Sources

- **[C1] Elm Electronics, ELM327 data sheet, version 2.3:** printed page 82, Figure 8, vehicle connector; printed page 84, Figure 9, reference circuit. Manufacturer PDF.
- **[C2] ScanTool.net / OBD Solutions, "Which OBD-II protocol is supported by my vehicle?":** connector illustrations, protocol assignments and ground/supply notes. Manufacturer guide. Its historical market/date statements are not used as current applicability guidance here.

OBD-II and virtual vehicle learning lab

Version 0.3.0 • 9 November 2026

1. What you are learning

This lab connects three ideas that are easy to confuse: a physical vehicle, the controllers that operate it, and the diagnostic interface used to inspect it. Start with one engine controller, then introduce a battery controller, a hybrid supervisor and brake controllers. The same plant and diagnostic boundaries can later host more detailed models or an external controller.

OBD-II is an onboard diagnostic framework concerned principally with emissions-related monitoring and reporting. An engine controller can report speed, temperature and faults through it. A vehicle contains many other controllers, and their manufacturer-specific data is not automatically available through the generic OBD-II PID set. A connector can carry both kinds of communication. The California Air Resources Board explains OBD-II's emissions-monitoring purpose; Toyota's training material separately identifies the engine, hybrid, battery and skid-control computers. [S1, T1]

In this package, the plant and controllers are executable teaching models. Diagnostic values come from the current simulation state rather than a fixed list of canned replies. That makes it possible to change the throttle, inject a sensor fault, observe a controller response and then read the resulting diagnostic data.

The boundary of this version

The engine stand is a mean-value model: it calculates average torque, speed, fuel use and temperature. It does not simulate individual combustion events. The hybrid is a **generic parallel hybrid teaching architecture**, with a lumped battery, an idealized variable-ratio engine path and simplified engine/motor power allocation. Its PHEV label describes the intended vehicle class; grid charging and charger communication are future modules. It is not a calibrated Toyota Prius, Prius Plug-in or Prius Prime simulation. ABS uses individual wheel dynamics; the yaw/ESP demonstration remains an approximation rather than a full lateral tire and suspension model.

The package does not run Toyota firmware. It does not connect to a vehicle, program an ECU, or reproduce a proprietary Toyota CAN database. The supplied examples are useful for learning, debugging interfaces and forming requirements; matching a particular vehicle requires measurements and model identification.

2. Your first learning session

Unzip the package, open a terminal in its top-level directory, and start the dashboard:

```
python3 run.py --serve
```

Use the local address printed by the program. On Windows, `python` may be the installed interpreter command instead of `python3`. The lab uses the Python standard library; it does not require a paid modeling tool.

Start with the learning book

The browser opens the Learning book. Choose a beginner chapter on the left, read the explanation and select **Start this experiment**. Use **Watch it run**, **Pause and inspect** and **One small step** to follow the result. The fault buttons disconnect a named input; the live notes explain what changes and why. Reading another chapter keeps the current experiment. The beginner handbook contains the same learning sequence and a glossary.

Open the schematic instruments when ready

Choose **Instruments** or **Open detailed instruments** to enter the focused diagnosis workspace. Its fixed block and circuit diagrams use the running model state; clicking a signal or lesson pin lets you inspect it and add a scope channel. This release does not implement free-form block editing, arbitrary rewiring, a SPICE solver or firmware execution.

Screen	Learning purpose
Task workspace	Follow the ECT fault task with a schematic, inspector, six-channel scope, diagnostic reply and controller timeline
Connectors	Inspect the J1962 vehicle-socket reference and the explicitly illustrative J-ECT/X1 lesson assignments
Signal list	Search 69 cataloged signals; inspect value, unit, source, validity and nominal update period; probe 67 numerical/digital channels
Scan tool	Send an application request and inspect its generated CAN/ISO-TP frames
Dashboard	Compare engine, battery, vehicle and braking responses in the other lessons

Start **Healthy baseline**, which runs the engine stand for 1 s from a selected 25°C initial coolant state. Choose **ECT electrical circuit**, then **Open circuit**, **+0.20 s**, **+0.30 s**, **Read DTC**, and **Repair & clear**. Compare physical temperature, sensed temperature and the ECU fallback. The workspace guide explains each observation; the connector reference gives orientation, provenance and pin assignments. J-ECT and X1 are invented teaching labels, not a Toyota wiring diagram.

Scope recordings are 100 ms apart. Invalid, substituted and stale finite values use dashed segments; missing or null values leave a gap. An open thermistor branch has a `null` resistance marked **invalid**, not a very large invented finite measurement. Use validity and the event timeline alongside the plotted numbers.

Continue through the vehicle lessons

Work through these repeatable scenarios in order:

1. Run **engine_warmup**. Watch engine speed, coolant temperature and fuel flow. Ask for `01 0C` and `01 05` in the diagnostic panel. Connect the physical value, encoded bytes and decoded result.
2. Run **ect_open**. Compare the true coolant temperature with the faulty sensor signal. Notice that a diagnostic report describes a detected electrical fault, not necessarily an overheated engine.
3. Run **phev_city**. Follow the path from driver request to engine/motor contribution, battery current and vehicle acceleration.
4. Run **regen_limit**. Observe the brake demand that the battery can accept and the demand that friction brakes must supply.

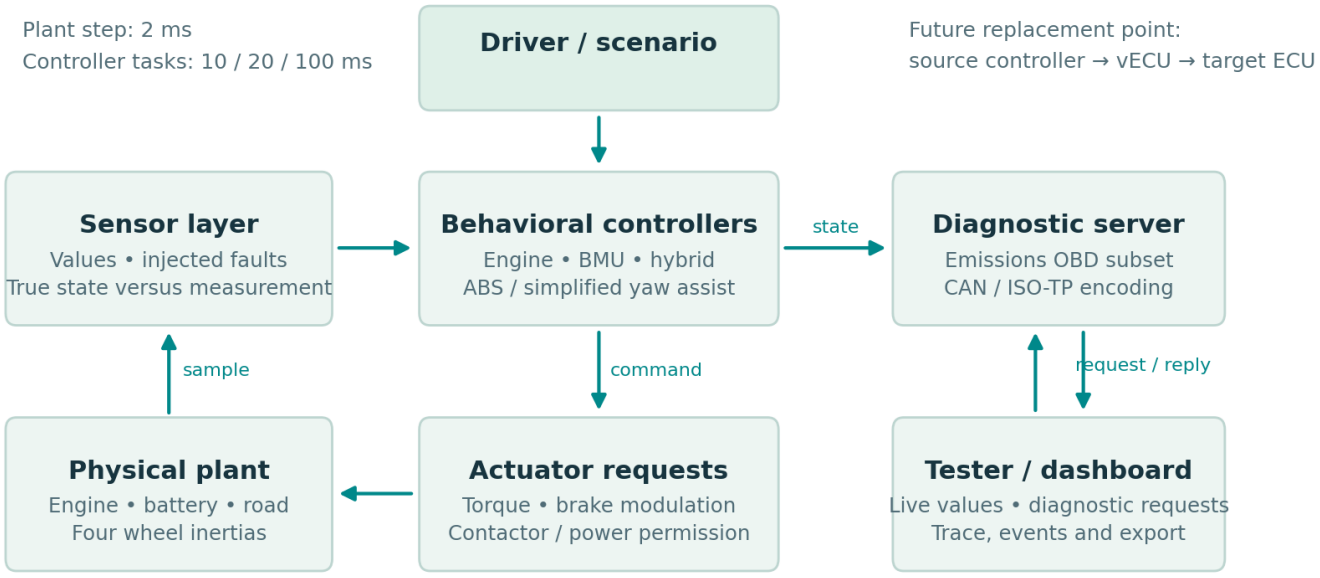
5. Run **abs_split_mu**. Compare four wheel speeds and slip values when left and right road friction differ.
6. Run **bmu_timeout** and **cell_low**. Examine why a controller can deny a request even when the driver continues pressing the accelerator.

The command line can reproduce a scenario and save data for inspection:

```
python3 run.py --scenario engine_warmup --duration 120 --output examples/my_warmup
python3 run.py --scenario phev_city --duration 60 --output examples/my_city
python3 -m unittest discover -s tests -v
```

Use the shipped example results as a starting point. Re-run after changing a parameter and compare the same quantities over the same time interval. A successful test checks the specified implementation behavior; it does not establish agreement with a real vehicle.

3. A controller does not create the physical result directly



The implemented feedback loop, diagnostic boundary, and future controller replacement point.

Consider a temperature reading. The engine has a physical temperature. A sensor and its electrical interface transform that temperature into a measurement. Software checks plausibility, filters or substitutes the reading, and makes a decision. The OBD server encodes a reported value. These are four separate quantities, even when they happen to agree.

Layer	Example	Question to ask
Plant	Crankshaft inertia, coolant heat capacity, tire forces	What changes according to physics?
Sensors	RPM estimate, coolant signal, pack current	What can the controller actually observe?
Controller	Idle control, battery power limit, brake-pressure request	What decision is made from those observations?
Actuators	Engine torque, inverter torque, contactor, brake torque	How does the decision affect the plant?

Layer	Example	Question to ask
Network and diagnostics	Periodic signals, request/response, DTCs	What is transmitted, at which time, and in which units?

The feedback loop repeats. A controller issues a bounded request; the plant responds; sensors sample that response; the next controller execution uses the new measurements. Holding one layer constant while changing another is a powerful experiment. For example, inject a coolant sensor fault while keeping actual coolant temperature unchanged.

A vocabulary for the virtual vehicle

Name	Role	Status in this lab
ECU	General term for an electronic control unit	Multiple conceptual controllers within one process
ECM / engine ECU	Engine torque, speed and thermal behavior; engine diagnostics	Simplified working controller and plant
BMU / BMS / battery ECU	Estimate battery state; calculate charge/discharge permissions	Working lumped-pack model and supervisory limits
PHEV-ECU / VCU / hybrid supervisor	Select propulsion mode; coordinate engine, motor and battery	Working generic supervisory model
Inverter and motor controller	Convert electrical power to torque and vice versa	Average power/torque model; electrical switching and current loops absent
ABS	Modulate braking when wheels approach lock	Working four-wheel longitudinal demonstration
ESP / ESC	Compare intended and actual vehicle rotation; influence torque and wheel braking	Approximate yaw-assist demonstration
OBC	Onboard charger for grid-to-battery energy transfer	Planned; no AC charge-port model in this release
DC/DC and 12 V supply	Power low-voltage loads and control electronics	Auxiliary power allowance; detailed low-voltage network planned
TCU / transmission controller	Gear, clutch or transmission coordination	Transmission represented by simplified ratios; no separate TCU
Gateway	Route messages between networks; control diagnostic access	Planned; current signal exchange is in process
BCM / body controller	Lighting, doors, wake/sleep and cabin functions	Planned
EPS	Electric power steering and steering assistance	Steering is an input; EPS motor and rack dynamics planned
SRS / restraint controller	Crash sensing and occupant restraint decisions	Outside this release; needs a separate specialized model
HVAC / thermal manager	Cabin, battery, engine and power-electronics heat management	Lumped engine/battery cooling only
Cluster / telematics / ADAS	Displays, external communication, driver assistance	Dashboard is a learning display; production functions planned

Real manufacturers distribute these functions differently. Several functions may share a computer; one function can also span multiple computers. The module names are architectural teaching boundaries, not a claim about the exact ECU inventory of a particular car.

4. What sensors and signals does each module need?

The following is a design checklist for extending the lab. Rates are **suggested educational starting points**, not OEM specifications. This release uses slower supervisory abstractions where electrical details are unnecessary. Its signal catalog carries a value, unit, snapshot timestamp, validity status, source and nominal update period for each registered signal. A snapshot timestamp is not a separately modeled sensor-acquisition or network-arrival timestamp; the BMU freshness model exposes its own message age.

Controller	Essential inputs	Outputs	Starting timing and failure behavior
Engine ECU	Crank/cam position, pedal and throttle position, coolant and intake temperature, MAF or MAP; later lambda, knock and fuel pressure	Torque/throttle request; later injection and ignition timing, fan and pump	Implemented average controller: 10 ms. Real crank events need angle/event scheduling. Reject implausible temperature; use a documented fallback and diagnostic.
BMU	Pack current A, pack voltage V, cell/block voltages V, temperatures °C; later isolation and contactor feedback	SOC %, available charge/discharge power, cooling demand, HV permission	Implemented supervisor: 100 ms. Fast hardware protection is a different layer. Stale or invalid limiting data must not grant unlimited power.
Hybrid supervisor	Driver request, vehicle speed, engine availability, BMU limits, gear/ready state, brake demand	Engine and motor requests, mode, start/stop coordination	Implemented supervisor: 20 ms. Hysteresis reduces rapid mode switching. Communication loss inhibits propulsion in the teaching policy.
Motor/inverter	Torque request, DC voltage/current, speed; later resolver angle, phase currents and device temperatures	Actual torque, power and limit flags; later PWM	Implemented average model follows plant steps. A future electrical model may need microsecond-scale integration and current-control scheduling.
ABS/ESC	Four wheel speeds, brake demand/pressure, steering angle, yaw rate; later lateral/longitudinal acceleration	Per-wheel brake modulation and requested torque reduction	Implemented brake control: 10 ms. A wheel-sensor failure must be represented explicitly, not silently replaced by a perfect reading.
Charger / DC/DC	Plug state, available AC power, HV and LV voltage/current, thermal state	Charging current, contactor requests, low-voltage regulation	Planned: choose supervisory and converter times separately. Inhibit traction during an accepted charging connection.
Gateway / body	Heartbeats, ignition/wake requests, network state, diagnostic sessions	Routed frames, wake/sleep and access state	Planned: model freshness deadlines and routing; message arrival is not proof of signal validity.

Choose signal units before choosing a bus

Use SI units inside a plant wherever practical: m/s, rad/s, N, N·m, A, V, W and seconds. Display conversions may use km/h, RPM, °C and kW. SOC is a fraction in equations and a percentage on the display. State which representation an interface uses.

For each future CAN signal, record identifier, direction, payload length, bit position, endianness, signedness, scale, offset, range, invalid code, period, timeout and any counter/checksum. A plausible CAN identifier is not a verified Toyota identifier. Keep synthetic teaching messages in a clearly named database; introduce real messages only with source evidence and versioned vehicle applicability.

5. The mathematics, one component at a time

These equations explain the model boundaries and provide hand-checkable exercises. Numeric values below are deliberately chosen examples. They are not measurements or calibration values from the supplied Toyota documents.

Implemented starting values

The hand exercises use round numbers so they can be checked independently. The executable plant starts from the following separate teaching assumptions, available in `obdii_lab/core.py`. Scenario-specific initial conditions can override the defaults.

Quantity	Implemented assumption	Interpretation
Vehicle mass; wheel radius	1,600 kg; 0.31 m	Generic vehicle, no Toyota fit
Wheel inertia; wheelbase	1.5 kg·m² per wheel; 2.70 m	Four longitudinal wheel states; simplified yaw reference
Battery capacity; internal resistance	50 Ah; 0.12 Ω	One lumped equivalent circuit
Open-circuit voltage	`300 + 100 × SOC` V	SOC is a 0-1 fraction; synthetic linear curve
Reported cell extrema	Aggregate pack voltage divided by 96, with a small spread	Illustrative signals, not 96 independent cell models
Nominal power ceilings	80 kW discharge; 40 kW charge	Further reduced by current, SOC, temperature and contactor state
Charge taper	90-98% SOC	Teaching thresholds, not a battery manufacturer’s limits
Motor/inverter efficiency	0.90 motoring; 0.88 regeneration	Constant averages, not measured efficiency maps
Precharge	First-order time constant 0.12 s; minimum 0.4 s and 95% of open-circuit voltage before closure	Behavioral sequence without switching arcs or a physical relay model
Engine stand inertia	0.28 kg·m²	Mean torque and virtual dynamometer load
Coolant thermal capacity	35,000 J/K	One heat-storage state with simplified cooling
ECT reference; pull-up	5.0 V; 2,200 Ω	Ideal reference shared with ADC; no wiring resistance or settling model
ECT thermistor	2,500 Ω at 25°C; beta 3,500 K	Selected NTC curve, not a Toyota sensor calibration

Quantity	Implemented assumption	Interpretation
ECT ADC	12 bits, counts 0-4095	Ideal rounding and reconstruction, without conversion noise

Read each configurable value's unit before changing it. Some controller constants remain in source; they are not all dashboard controls. Increasing the UI playback speed advances more simulated time and does not change these physical parameters.

Precharge voltage is a behavioral timing state: capacitor/resistor energy and low-voltage starter energy are not included in the battery ledger. The energy counters support component-level checks, including battery chemical power = terminal power + internal resistive loss. They do not establish complete vehicle energy closure.

5.1 Engine speed and mean torque

Rotational dynamics follow:

$$J\frac{d\omega}{dt} = T_{engine} - T_{load} - T_{friction}, \qquad RPM = \omega\frac{60}{2\pi}.$$

`J` is rotational inertia in kg·m². A controller cannot change engine speed instantly when inertia is represented. Suppose `J = 0.20`, engine torque is `30 N·m`, and load plus friction is `10 N·m`. Acceleration is `100 rad/s²`. During `0.01 s`, speed increases by `1 rad/s`, or about `9.55 RPM`.

Mechanical power is `P = Tω`. At `100 N·m` and `2,000 RPM`, shaft power is `20.94 kW`. Fuel power must exceed useful shaft power. If an illustrative efficiency is `0.30`, fuel chemical power would be `69.81 kW`. At `8.9 kWh/L`, that corresponds to `7.84 L/h`. Idle is a special case: useful shaft output can be near zero while fuel is still consumed to overcome friction and maintain combustion.

A simplified intake estimate can depend on displacement, air density, volumetric efficiency and engine speed. A four-stroke cylinder fills once every two revolutions, so total ideal intake volume flow is proportional to displacement times `RPM/120` when RPM is converted to revolutions per second through that factor. Real intake flow also depends on manifold pressure, valve timing and transients; an educational MAF estimate is not an identified airflow model.

5.2 Temperature requires energy storage

Use a lumped thermal model:

$$C_{th}\frac{dT}{dt} = \dot{Q}_{in} - h(T - T_{ambient}).$$

If heat capacity is `100,000 J/K` and net heating is `10,000 W`, temperature initially rises `0.10 °C/s`. A `60 °C` rise would take at least ten minutes if that net heat stayed constant. A realistic warm-up demonstration should therefore show gradual heating. Do not accelerate physical heating merely because the display is playing at a faster wall-clock speed.

This one-node model cannot resolve coolant circulation, thermostat opening, cylinder-head hot spots or catalyst light-off. Add those states when the experiment needs them. Keep an ECT electrical fault separate from genuine overheating.

5.2a From coolant temperature to an electrical measurement

A thermistor changes resistance with temperature. A fixed pull-up and an NTC thermistor to return form a divider; increasing temperature lowers the NTC resistance and the input voltage. A shared divider/ADC reference makes the measurement ratiometric. Texas Instruments documents this circuit principle and the beta-model conversion; its component choices and acquisition circuit are not the lab's chosen parameters. [S12]

For absolute temperature `T` in kelvin, the implemented NTC law and divider are:

$$R(T) = R_{25}\exp\left[B\left(\frac{1}{T} - \frac{1}{298.15}\right)\right], \qquad V_n = V_{ref}\frac{R(T)}{R_p + R(T)}.$$

The default values are `R25 = 2500 Ω`, `B = 3500 K`, `Rp = 2200 Ω` and `Vref = 5 V`. The branch current is `I = Vref / (Rp + R)`. A 12-bit ADC transforms the physical voltage into an integer count and reconstructs a measured voltage:

$$N = \text{round}\left(4095\frac{V_n}{V_{ref}}\right), \qquad V_m = V_{ref}\frac{N}{4095}.$$

At exactly 25°C, `R = 2500 Ω`, so `Vn = 2.659574 V`, branch current is `1.06383 mA`, `N = 2178` and `Vm = 2.659341 V`. The inverse calculation uses `Rm = Rp × Vm / (Vref − Vm)` and `Tc = 1 / [1/298.15 + ln(Rm/R25)/B] − 273.15`. It returns about **25.00477°C**. That small difference follows from ideal ADC quantization; it is not thermal change or a drop along the wire.

Opening the modeled sensor path makes current zero and brings the ECU/harness node to **5.000 V**, with ADC count **4095**. The branch resistance is `null`, marked invalid, because no finite resistance is reported. Validity checks prevent the inverse formula from dividing by zero. The lab shows **−40°C** for the invalid sensor report. At 0.20 s of high input it substitutes **80°C** for control; those are separate signals, neither a sudden physical temperature change.

The high-input threshold is 99% of reference, or 4.95 V at defaults. P0118 becomes pending at 0.20 s and confirmed with MIL at 0.50 s. These are accelerated teaching thresholds, not statutory trip logic or OEM calibration. The ideal model omits connector resistance, sensor lag, ADC sample/hold settling, noise and supply transients. Version 0.2 replaces the previous linear sensor placeholder and 4.98 V open value with this divider/ADC model.

5.3 Vehicle acceleration and road load

For forward longitudinal motion:

$$m\dot{v} = F_{drive} - F_{brake} - F_{roll} - F_{aero} - mg\sin\theta,$$

$$F_{roll} = C_{rr}mg\cos\theta, \qquad F_{aero} = \frac{1}{2}\rho C_d A v^2.$$

At `m = 1,500 kg`, `Crr = 0.01`, `ρ = 1.225 kg/m³`, `CdA = 0.60 m²`, and `v = 20 m/s` on level ground, rolling resistance is `147.15 N` and aerodynamic drag is `147 N`. Steady-speed wheel power is about `5.883 kW`. A 5% uphill grade adds approximately `735 N` at small angles, or another `14.7 kW` at that speed. Climbing can therefore dominate aerodynamic drag at moderate speeds.

Wheel torque and force relate through `T = Fr`. With `r = 0.30 m`, a `3,000 N` drive force requires `900 N·m` at the driven wheels. Gear ratios and efficiency connect wheel torque to motor/engine shaft torque. A pure power

divided by speed calculation becomes singular at zero speed; use a torque limit for launch and transition to a power limit as speed increases.

5.4 Battery equivalent circuit and SOC

Adopt **positive current and positive battery terminal power for discharge**. For a simple open-circuit voltage and internal resistance model:

$$V_t = V_{oc} - IR, \qquad P_t = (V_{oc} - IR)I.$$

For a feasible requested power, the physically useful lower-current branch is:

$$I = \frac{V_{oc} - \sqrt{V_{oc}^2 - 4RP_t}}{2R}.$$

The expression under the square root must be nonnegative. Current, voltage, temperature and SOC limits usually impose tighter bounds than this mathematical maximum. Clip the *allowed power request* before updating energy; clipping SOC afterward does not repair an impossible power flow.

With `Voc = 350 V`, `R = 0.15 Ω` and discharge power `20 kW`, current is approximately `58.62 A`; terminal voltage is `341.21 V`. Internal heating is `I²R ≈ 515 W`. Chemical battery power is terminal power plus this loss, about `20.515 kW`.

Coulomb counting for capacity `Q_Ah` is:

$$SOC_{k+1} = SOC_k - \frac{I\Delta t}{3600 Q_{Ah}}.$$

At `60 A` for `10 s` with `40 Ah`, SOC falls by `0.004167`, or **0.4167 percentage points**. A fall from 60.0% to 59.5833% is not a 0.4167% relative reduction. During charging current is negative, so SOC rises. A future estimator should add sensor bias, temperature dependence, relaxation, capacity aging and an initial-SOC correction method.

5.5 Brake blending and energy recovery

The driver asks for deceleration. The supervisor allocates some braking to generation, subject to motor, tire and battery constraints. Friction brakes provide the remainder:

$$P_{brake,requested} = P_{regen,mechanical} + P_{friction}.$$

Regen mechanical power and battery charging power are different. If the generator/inverter path is `90%` efficient and the battery permits `10 kW` of terminal charging, it can accept at most about `11.11 kW` of mechanical recovery through that path, before considering auxiliary loads and other constraints. For a `30 kW` mechanical brake demand, at least `18.89 kW` becomes friction heat in this example.

Available kinetic energy is `½mv²`. A `1,500 kg` car at `50 km/h` contains about `144.7 kJ`, or `0.0402 kWh`, of translational kinetic energy. Only part can reach the battery. Aerodynamic drag, rolling resistance, friction brakes, machine losses and battery losses all consume energy. SOC near its upper limit, a hot battery, low motor speed or ABS intervention can reduce recovery. Braking must still respond to the driver’s demand when charge acceptance disappears.

5.6 Wheel slip, ABS and approximate yaw control

A useful braking-slip definition is:

$$\lambda = \frac{v - r\omega_w}{\max(v, v_\varepsilon)}.$$

When vehicle speed is `20 m/s` and wheel peripheral speed is `16 m/s`, slip is `0.20`. A locked wheel gives slip near `1` while the vehicle is still moving. Near rest, the denominator safeguard and a low-speed ABS cutoff are essential.

Wheel dynamics include brake torque and tire reaction:

$$J_w\dot{\omega}_w = T_{drive} - T_{brake} - rF_x.$$

The sign of `Fx` follows the chosen wheel/vehicle convention. In braking, the ground reaction that slows the vehicle also tends to spin the wheel back toward rolling speed. The tire-force limit depends on friction and normal load. ABS reduces pressure when slip grows too large and reapplies it after wheel recovery. The best slip target varies with the tire and surface; a fixed teaching threshold is not a universal optimum.

A small-angle bicycle reference for desired yaw is `r_ref ≈ v tan(δ)/L`. With speed `15 m/s`, road-wheel steering angle `0.03 rad`, and wheelbase `2.7 m`, the reference is `0.1667 rad/s`, about `9.55°/s`. The road-wheel angle is not necessarily the steering-wheel angle. A tire-friction bound also limits physically achievable yaw. Production ESC uses wheel speed, steering and inertial measurements and can reduce propulsion or brake individual wheels. [S5]

This release can illustrate a yaw error and an intervention flag, but cannot validate cornering stability, spin prevention or stopping-distance performance. Those questions require a calibrated lateral tire model, load transfer, brake hydraulics and sensor/actuator delays.

6. Four different meanings of “cycle”

Cycle	Meaning	What this release does
Physical integration step	The interval used to advance differential equations	Fixed 2 ms plant steps
Controller task cycle	How often software reads inputs and updates outputs	Engine/brake 10 ms, hybrid 20 ms, BMU 100 ms
Driving cycle	A time series of driver or speed requests and environmental conditions	Educational warm-up, city, fault and braking scripts
Engine combustion cycle	Intake, compression, power and exhaust events	Averaged; no cylinder pressure or angle-resolved combustion

At `2,000 RPM`, a four-stroke cylinder completes its combustion cycle about `16.67` times per second. That has no direct relationship to a 10 ms diagnostic refresh or a 60-second driving example. Display refresh, simulation time, bus timestamps and controller scheduling should remain distinguishable.

Integer task ticks make reproducible scheduling easier. The simulator applies scenario inputs and persistent user overrides, runs due BMU work, then hybrid supervision, then engine diagnostics and brake control, and finally advances the plant. It increments simulated time, records state-transition events and logs every 100 ms. Controller measurements consequently refer to the plant state available before that integration step.

The precise order is part of the model contract. When adding another solver or external ECU, define who owns simulated time and what happens at a missed deadline.

These examples are not WLTP, EPA certification or OBD monitor-completion procedures. A future imported speed trace needs its source, units, time grid, driver-tracking controller and measured tracking error recorded.

7. Working cases and what each proves

Case	Setup and observation	Learning result and important limit
`engine_warmup`	Run the engine stand from a cold state; inspect RPM, fuel and coolant	Links torque control, thermal inertia and OBD data; 120 seconds is a partial warm-up, without catalyst chemistry
`ect_open`	Inject an open ECT signal after normal operation begins	Separates physical temperature, sensor failure, fallback and P0118; short teaching debounce is not the real regulatory trip logic
`phev_city`	Repeat acceleration, cruise and braking requests	Demonstrates propulsion allocation, battery discharge and regeneration; demonstration can exceed urban speeds, no Toyota eCVT or calibrated city cycle
`low_soc`	Start with reduced SOC and request propulsion	Shows reserve logic and engine use; engine engagement does not imply a detailed charge-sustaining optimization
`abs_split_mu`	Brake from speed with different left/right road friction	Shows why wheel behavior differs; no claim of a vehicle-validated stopping distance
`regen_limit`	Begin with restricted battery charge acceptance	Demonstrates friction replacement when regeneration is limited; electrical converter waveforms absent
`bmu_timeout`	Stop fresh BMU updates while continuing driver demand	Shows freshness checks, inhibition and event logging; no physical CAN arbitration or error frames
`cell_low`	Inject a low-cell condition into the battery supervision	Shows that a pack-level average can hide a limiting element; individual electrochemical cells are not modeled

For each case, inspect both the time trace and event log. A single final value can conceal a transient violation. CSV export provides selected numerical traces; JSON also preserves events, final sensor state, energy counters and diagnostic transcripts. A command-line or dashboard input override persists over the scenario script until the simulation is reset. Record the input history, parameters, initial conditions and software version so that a result can be repeated.

Try four useful comparisons: ABS enabled versus disabled on the same surface; normal versus high SOC for the same brake request; normal versus lost BMU updates; and the same engine load at two ambient temperatures. Change only one condition at a time until you can explain the result. If the result surprises you, first check units and signs, then saturation and validity rules, then scheduling.

8. How the Toyota material fits

The user's supplied Course 071 explains a more specific architecture: engine, MG1 and MG2 are mechanically coupled through a planetary unit, while the HV ECU coordinates the ECM, inverter, battery ECU and skid-control ECU. Its Gen2 example has a 201.6 V NiMH pack. Battery monitoring and a resistor-assisted main-relay sequence are useful architectural references. [T1, T2]

For that planetary set, the ideal speed relationship is:

$$(N_s + N_r)\omega_{carrier} = N_s\omega_{sun} + N_r\omega_{ring}.$$

With `Ns = 30` and `Nr = 78`, engine/carrier speed `1,500 RPM` and ring/MG2 speed `3,000 RPM`, MG1/sun speed must be `−2,400 RPM`. A negative speed is a valid rotational direction; it does not by itself identify motor or generator power. Power also depends on torque sign. This kinematic exercise is an extension target, not an implemented drivetrain in this lab. [T1]

The existing `MODEL_NOTES.txt` already separates a Prius II HEV baseline from a hypothetical plug-in extension and lists its own assumptions. Preserve that distinction when comparing projects: an earlier 8.8 kWh hypothetical pack is not evidence of a factory Gen2 PHEV. The existing firmware guide also distinguishes static engine-image research from an authenticated, executable HV or battery ECU firmware model. Those prior artifacts are reference material; this package does not import or execute their binaries. [U1, U2]

9. From a lesson to a credible virtual vehicle

Add fidelity because an experiment requires it. For OBD encoding lessons, a stable mean-value engine is enough. For torque-response testing, add actuator delay and sensor filtering. For battery SOC estimation, use dynamic voltage relaxation and sensor errors. For ABS performance, add calibrated tires, load transfer, road geometry and hydraulics. For inverter firmware, model ADC sampling, PWM timing, current loops and the motor's electrical equations.

Keep a requirement table with an input condition, expected response, deadline, tolerance and evidence. For example: "When BMU data age exceeds the configured timeout, the hybrid supervisor shall stop issuing positive motor torque within one supervisor period." The test must exercise the timing boundary, not merely confirm that the event text exists.

Use three kinds of checks together: physical invariants (energy and force balance), behavioral requirements (fault response and state transitions), and comparison with measured data (accuracy). Record fuel chemical energy, battery terminal energy, battery internal losses, useful mechanical energy, friction braking and auxiliary consumption at declared boundaries. A numerically closed ledger is necessary, but it cannot prove that invented efficiencies or thresholds match a real car.

The workspace guide explains the interactive task and instruments, and the connector reference distinguishes standard DLC roles from invented lesson pins. The firmware and virtual-vehicle integration path is described in **FIRMWARE_ROADMAP.md**. The exact diagnostic subset and frame examples are in **PROTOCOL.md**. **EXISTING_BASES.md** explains the existing software selected for reuse and the optional virtual-CAN adapter. **SOURCES.md** records the references and separates manufacturer evidence from teaching assumptions.

Worked cases: observations you can reproduce

Version 0.3.0 · 9 November 2026

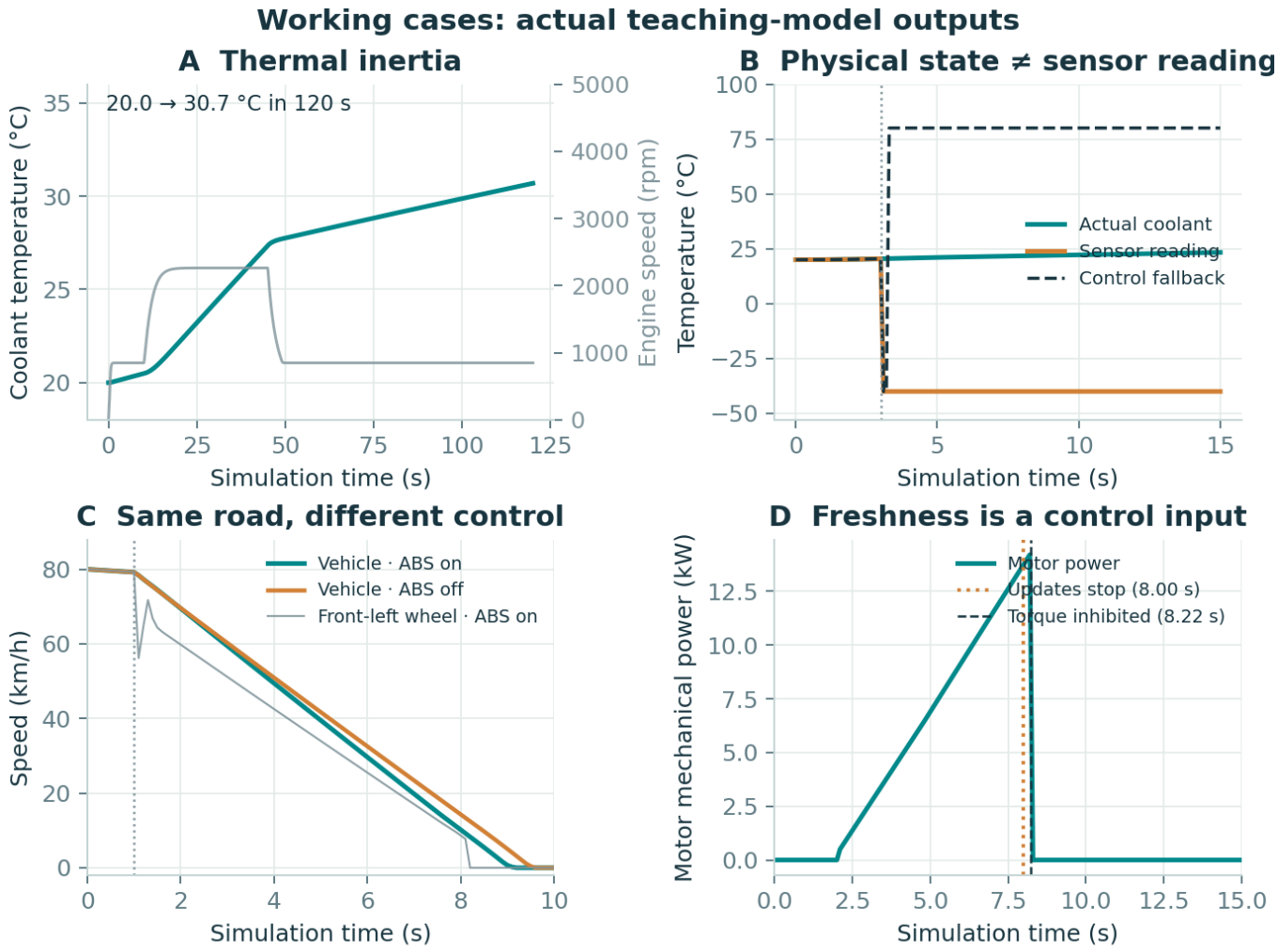
These results were computed by the supplied simulator, using its default generic teaching parameters. They are **synthetic results**, not Toyota measurements or calibration. The example files are generated by the same core used by the dashboard and command-line runner. Each run has a CSV time series, a JSON file containing parameters, states, events and final diagnostic requests, and a human-readable CAN transcript under `examples/runs/`.

From the unpacked project directory, regenerate everything with:

```
python3 tools/generate_examples.py
```

Time-series samples are 100 ms apart; the plant step is 2 ms. A controller change at an exact tick can first appear in the following saved sample. Event timestamps are the better source for fault-detection times. A “stop below 0.5 km/h” and a zero-speed sample are deliberately separate summary fields.

1. Start a single engine ECU and follow the heat



Synthetic results, not measured Toyota data. See the worked cases for parameters and numerical limits.

Actual simulator traces for thermal inertia, sensor failure, split-friction ABS and BMU freshness.

Hypothesis: combustion produces torque immediately after cranking, but coolant has thermal inertia. The dyno stand stays stationary even when engine speed rises.

```
python3 run.py --scenario engine_warmup --output examples/my_warmup
```

The engine starts from 0 rpm, passes through CRANKING and RUNNING, and reaches a sampled peak of **2263.3 rpm** as the script changes accelerator position. At 120 s the actual coolant has risen from 20.0°C to **30.68°C**, with **0.05339 L** of modeled fuel used. Vehicle speed remains zero because this profile is a stand.

This is partial warm-up. Do not expect operating temperature after every two-minute run. Compare ECU crank speed, mass airflow and coolant PIDs against the physical state. Extend the duration or change the manual load to explore the heat balance; the extra time continues with the script’s final idle input.

2. Disconnect the coolant-temperature sensor

Hypothesis: a sensor reading can be wrong while the physical quantity remains normal; clearing diagnostic memory does not repair that sensor.

```
python3 run.py --scenario ect_open --output examples/my_ect_fault
```

At 3.00 s, the ECT circuit becomes open. Its simulated pull-up input is **5.000 V**, with a full-scale **4095** ADC count and zero branch current. The branch resistance is `null`, marked invalid; no finite resistance is reported. The sensor temperature display is **−40°C**, with invalid measurement status. After 200 ms the teaching ECU marks P0118 pending, uses an 80°C fallback estimate, and applies a 65% torque multiplier. At **3.50 s**, P0118 becomes confirmed and the MIL turns on. The final physical coolant is **23.32°C**, while the reported sensor remains **-40°C**. These are distinct signals.

In the final transcript, `01 05` returns `41 05 00`; the temperature conversion is $0 - 40 = -40$°C. `03` reports P0118, encoded as bytes `01 18`. Use the dashboard's clear button, continue time, and observe the fault returning. These 200/500 ms rules are an accelerated lesson, not regulatory emissions confirmation logic.

Inspect the same cause in the new task workspace

The six task buttons provide a separate interactive setup: **Healthy baseline** runs 1 s from a selected 25°C coolant initial condition, then **Open circuit**, **+0.20 s**, **+0.30 s**, **Read DTC**, and **Repair & clear** expose each transition. This task begins near 25°C; the scripted `ect_open` recording above retains its 20°C initial condition and its fault at 3.00 s.

Choose **ECT electrical circuit**, then probe the physical node voltage, ADC count, reported temperature and control temperature. At exactly 25°C the synthetic 2,500 Ω NTC and 2,200 Ω pull-up give `5 × 2500 / 4700 = 2.659574 V`. The 12-bit ADC gives count 2178 and reconstructs 2.659341 V. Opening the path separates true coolant temperature from the invalid sensor report and later 80°C fallback.

Invalid, substituted and stale finite scope values use dashed segments. Null resistance leaves a gap. Recorded samples are 100 ms apart, so inspect the event timeline for the monitor's exact transition time. The workspace guide and connector reference explain the five screens, measurement boundaries and illustrative harness mapping. Clearing the code while the input remains open still allows it to return after the teaching debounce.

3. Follow power through EV, hybrid and braking states

Hypothesis: the supervisor changes sources and braking allocation while the same physical vehicle continues moving.

```
python3 run.py --scenario phev_city --output examples/my_phev_cycle
```

The DC link precharges before the contactor closes; torque is inhibited during precharge. The first launch is electric, the first braking event recovers energy, and the later high-demand launch also uses the engine. The vehicle reaches **102.00 km/h**, stops twice, and ends at zero speed after 60 s. SOC changes from 75.0% to **74.461%**; gross battery discharge is **0.16177 kWh**, and gross charge is **0.06572 kWh**. Gross charge and net SOC change are different quantities because driving and auxiliaries consume energy during the same run.

This is an educational sequence with deliberately visible transitions, not an EPA, WLTP, Toyota, or standardized urban drive cycle. The transmission is a generic parallel-hybrid approximation, not a Prius planetary gearset.

4. Begin with low SOC

Hypothesis: the engine can support propulsion and a charging request when the supervisor enters charge-sustain operation.

```
python3 run.py --scenario low_soc --output examples/my_low_soc
```

The pack begins at 22.0% SOC and ends at **22.420%** after 45 s. The supervisor requests up to 5 kW of generator shaft power; the sampled maximum is **5.00 kW**. Mode is CHARGE SUSTAIN during traction. Battery current is negative while net charging, and engine fuel use is **0.09741 L**. The narrow run increases SOC by only **0.420 percentage points**; it is not a complete battery recharge. Generator output is limited when the battery cannot accept charge.

5. Compare ABS on the same split-friction surface

Hypothesis: per-wheel pressure modulation reduces sustained locking in this specified model. Keep every other input identical for the comparison.

```
python3 run.py --scenario abs_split_mu --output examples/my_abs
python3 run.py --scenario abs_split_mu --set abs_enabled=false --output examples/my_no_abs
```

Both runs start at 80 km/h, coast for 1 s, then apply 85% braking. Left-road friction is 0.20 and right-road friction is 0.80. The shared first-second coast covers **22.11 m**.

Observation	ABS enabled	ABS disabled
Total distance from time zero	110.32 m	115.51 m
Distance after brake command at 1 s	88.21 m	93.40 m
First saved speed below 0.5 km/h	9.1 s	9.5 s
ABS active time	7.07 s	0.00 s

Inspect the four slip and pressure traces; with ABS disabled, wheels reach locking slip. These distances apply to this toy tire curve, pressure controller and fixed conditions. They do not establish real-vehicle stopping performance. The associated yaw-brake assistance is reduced-order and omits lateral tire and body-sideslip states.

6. Ask a nearly full battery to accept regeneration

Hypothesis: restricted charge acceptance moves braking work to friction brakes.

```
python3 run.py --scenario regen_limit --output examples/my_full_pack_braking
python3 run.py --scenario regen_limit --config examples/normal_soc.json --output
  examples/my_normal_pack_braking
```

Both vehicles start at 60 km/h and apply the same 30% brake command after 1 s. Only initial SOC changes: 97.5% versus 75.0%.

Energy over the 35 s experiment	97.5% starting SOC	75.0% starting SOC
Gross battery charge	0.00856 kWh	0.02662 kWh
Mechanical regeneration	0.01157 kWh	0.03209 kWh

Energy over the 35 s experiment	97.5% starting SOC	75.0% starting SOC
Friction-brake dissipation	0.04335 kWh	0.02249 kWh
Total distance	120.60 m	120.60 m

Charge power tapers between 90% and 98% SOC. Mechanical regeneration exceeds stored battery energy because the motor/inverter conversion is lossy and auxiliaries still consume power. The stopping distances remain similar because friction braking fills the request when regeneration is limited.

7. Remove the BMU heartbeat

Hypothesis: a supervisor must distinguish stale data from a healthy unchanged measurement.

```
python3 run.py --scenario bmu_timeout --output examples/my_missing_bmu
```

Messages stop at 8.00 s. The last message arrived at 7.90 s; the 20 ms supervisor therefore detects message age greater than 300 ms at **8.22 s**. It records the local `BMU_STATUS_TIMEOUT` teaching label, inhibits torque, and opens the traction path. The car continues coasting and ends at **19.76 km/h**. A fault does not magically remove kinetic energy. No generic emissions MIL is asserted for this local network fault.

8. Force a low cell-voltage input

Hypothesis: persistent out-of-range battery sensing causes a latched traction shutdown rather than continued torque production.

```
python3 run.py --scenario cell_low --output examples/my_low_cell
```

At 8.00 s one measured cell channel is forced to 2.30 V. The BMU threshold is 2.80 V for 200 ms, so shutdown occurs at **8.20 s**. Final contactor status is FAULT, and battery current and traction output are zero. The stored label is `BMU_CELL_UNDERVOLTAGE`; it is explicitly a teaching identifier, not a claimed Toyota trouble code. Ordinary cells are aggregate pack/96 readings with a nominal spread; this does not model 96 independent cell electrochemistry states. Remove the injection and cycle ignition to reattempt readiness.

Read the recordings critically

`examples/RESULTS.csv` and `RESULTS.json` provide the comparison values above. Each `.trace.txt` contains requests made **at the end of its run**, not requests polled continuously throughout the CSV. Thus a healthy engine-speed response can be zero at the end of a PHEV stop while earlier records show combustion running.

The battery ledger checks chemical energy = terminal energy + resistive heat. This is a check on the modeled battery equations, not proof of complete vehicle energy closure: precharge resistor/capacitor energy, detailed 12 V loads, lateral motion, full engine maps and several other losses are outside this first model.

OBD-II and ISO-TP: the diagnostic lesson

1. What the interface does

OBD-II is the diagnostic window into an ECU. It does not calculate vehicle motion, manage the battery contactors, or execute the engine controller. In this lab the vehicle model produces a sensor snapshot; the diagnostic server encodes selected values into request/response messages. SAE J1979 describes emissions-related diagnostic services and distinguishes them from the transport and physical layers [R1].

The separation is:

Layer	Example in this lab	What it means
Physical process	Engine actually turning at 2,000 rpm	A mathematical state of the simulated plant
Sensor/controller	Crank-speed estimate and ECU state	The ECU's view of that process
Diagnostic application	Request `01 0C`; reply `41 0C 1F 40`	Ask for engine speed and encode the result
Transport	ISO-TP single frame	Add a length/control byte and, if necessary, split a large reply
Data link	CAN identifier 7E8, eight data bytes	Identify the sender/message on the configured network
Electrical network	CAN transceivers and wiring	Not simulated in this release

Implemented scope: one emissions ECU, Classical CAN, 11-bit identifiers, normal ISO-TP addressing, and a small set of services. The configured bitrate is 500 kbit/s, but the trace is not a timing or bus-load simulation. No physical adapter is opened. This is a learning implementation, not an ISO/SAE conformance claim.

An optional working integration, `examples/python_can_demo.py`, reuses the existing **python-can 4.6.1 VirtualBus** transport. Two same-process endpoints exchange live requests and responses through real library queues. The example validates flow control before sending consecutive frames, and reassembles the reply. It is a bounded cooperative lesson rather than a complete asynchronous ECU server. See `EXISTING_BASES.md` for installation, reuse choices, and the next integration stages.

Toyota boundary: the identifiers and data values below describe this lab configuration. They are not a claim that every Toyota, Prius generation, powertrain ECU, battery ECU, or gateway uses the same arrangement. Generic emissions requests do not automatically expose every vehicle subsystem. An actual vehicle needs the correct model/year/network specification and supported diagnostic services.

2. Addresses and supported requests

Address	Role in the lab
0x7DF	Functional emissions request: addressed to potential responders
0x7E0	Physical request to this lab’s engine ECU
0x7E8	Response from that ECU

These are conventional 11-bit OBD-on-CAN examples documented by Elm Electronics [R3]. A real functional query can receive more than one ECU reply. This release intentionally creates one responder. Requests to other identifiers produce no response.

Enter application bytes such as `01 0C`, without the CAN identifier, ISO-TP length byte, or padding. This interface does not implement an ELM327 serial command interpreter: `AT Z`, `AT SP`, and other AT commands are not accepted.

Application request	Reply content	Lab behavior
01 00	Supported Mode 01 PID bitmap	41 00 98 1B 80 00
01 01	MIL, DTC count, limited monitor flags	Only a simplified comprehensive-component monitor is advertised
01 04	Calculated engine load	Model’s load proxy
01 05	Engine coolant temperature	ECU sensor reading, which can differ from true plant temperature
01 0C	Engine speed	Quarter-rpm encoding
01 0D	Vehicle speed	Whole km/h encoding
01 0F	Intake-air temperature	Ambient-temperature proxy in this model
01 10	Mass air flow	Model’s air-flow estimate
01 11	Throttle position	Model’s throttle signal
03	Confirmed emissions DTCs	CAN-format count followed by two-byte codes
07	Pending emissions DTCs	Same count-and-code structure
04	Clear simulated emissions diagnostic memory	Requires a connected clearing callback; an active fault is detected again
09 00	Supported Mode 09 bitmap	49 00 40 00 00 00
09 02	Synthetic VIN field	LAB00000000000001, exactly 17 characters

The synthetic VIN is a teaching identifier, not an OEM-issued or registration-valid identity. No real user’s VIN is included.

Unsupported services/PIDs and unsupported request shapes produce no reply. This intentionally avoids inventing a UDS negative response to a functional J1979 request. Modes 02, 06, 08, and 0A, multi-PID requests, ECU discovery, and manufacturer-specific requests are outside this release. Although the core can retain a first-fault snapshot for teaching, Mode 02 freeze-frame transport is not implemented.

3. Work through the bytes

Example A - 2,000 rpm

The requested mode is `01` and the engine-speed PID is `0C`.

```
Application request: 01 0C
Application reply:   41 0C 1F 40
```

The positive-response mode is the request mode plus hexadecimal `40`. The two data bytes are A=`1F` (31 decimal), B=`40` (64 decimal).

```
rpm = (256 × A + B) / 4
    = (256 × 31 + 64) / 4
    = 2,000
```

The quarter-rpm representation is described in the manufacturer’s diagnostic interpreter datasheet [R3]. The exact numerical example above is calculated for this lab.

Direction	CAN ID	Eight data bytes	Interpretation
Tester → ECU	7DF	`02 01 0C 00 00 00 00 00`	02 means two application bytes follow
ECU → tester	7E8	`04 41 0C 1F 40 00 00 00`	04 means four application bytes follow

The zero bytes at the end are padding. They are not additional sensor values.

Example B - coolant at 90 °C, then an open sensor

```
T_coolant = A − 40
A = 90 + 40 = 130 decimal = 82 hexadecimal
Reply = 41 05 82
```

An open NTC sensor circuit can be represented as a high signal voltage and an implausibly cold sensed temperature. In the lab’s fault case:

Quantity	Illustrative value	Meaning
`coolant_c`	90 °C	True mathematical plant temperature
`sensors.ect_c`	−40 °C	Faulted sensor reading
`sensors.ect_used_c`	80 °C after detection	Toy ECU fallback for control
`01 05` reply	`41 05 00`	Encoded faulted sensor reading: 0 − 40 = −40 °C
Emissions fault memory	P0118 after debounce	The demonstration’s engine-coolant high-input fault

The specific fallback temperature, detection delay, and reported fault value are teaching choices. Production diagnostics may report different substituted values and require different enable conditions and drive cycles. They must be verified against the actual ECU specification.

Example C - supported PIDs

`01 00` returns a 32-bit bitmap, not a list of 32 values. The most significant bit represents PID 01; the least significant bit represents PID 20. A set bit advertises support. The lab advertises precisely the requests in the table above, and its tests compare every bit against the implemented service. No higher PID range is advertised.

Example D - diagnostic trouble codes

For Classical-CAN OBD, the Mode 03 payload includes a count byte before the encoded codes [R3]:

Payload bytes	Meaning
`43`	Positive response to Mode 03
`01`	One DTC follows
`01 18`	P0118 in its two-byte representation

The four families use two high bits: powertrain `P`, chassis `C`, body `B`, and network `U`. The next two bits encode the first numeric digit (0-3), and the remaining 12 bits encode the last three hexadecimal digits. Examples computed by this lab:

Human-readable DTC	Two encoded bytes
`P0118`	`01 18`
`P0300`	`03 00`
`C1234`	`52 34`
`U0100`	`C1 00`

An empty confirmed-DTC list returns `43 00`. A code’s valid binary format does not prove that its definition is assigned or applicable to a particular ECU. The simulated module fault list is separate from the emissions `dtcs` list. ABS/ESP/BMU warnings should not be indiscriminately inserted into the engine ECU’s generic Mode 03 reply.

Example E - clearing does not repair the sensor

1. Run the ECT open-circuit scenario until P0118 appears.
2. Read `03`; the response contains P0118.
3. Send `04`; the simulator clears its emissions diagnostic memory and returns `44`.
4. The underlying injected fault remains active. Advance simulation time; the monitor can detect it again.
5. Remove the injected fault and clear the memory to observe a fault-free monitor run.

The toy monitor uses compressed persistence times so the lesson is visible quickly. It does not reproduce regulatory confirmation, healing, permanent-code, warm-up-cycle, or readiness rules. Real Mode 04 has broader semantics than the implemented teaching memory [R3].

4. A longer message: VIN over ISO-TP

ISO-TP distinguishes physical and functional addressing and supports single-frame and multi-frame transfers. In a multi-frame transfer, the receiver sends flow control that specifies block size and minimum separation time before the sender continues [R2].

The 20-byte application payload here consists of `49 02 01` followed by the 17 VIN characters. It cannot fit into the seven application bytes available in one normally addressed Classical CAN frame.

Step	CAN ID	Frame bytes	Meaning
1	7DF	02 09 02 00 00 00 00 00	Functional VIN request
2	7E8	10 14 49 02 01 4C 41 42	First frame: payload length 014 hex = 20 bytes; begins LAB
3	7E0	30 00 00 00 00 00 00 00	Flow control to the responding ECU’s physical address
4	7E8	21 30 30 30 30 30 30 30	Consecutive frame, sequence number 1
5	7E8	22 30 30 30 30 30 30 31	Consecutive frame, sequence number 2

`30` is Continue To Send. The chosen block size is zero (no intermediate block limit), and STmin is zero. The lesson trace inserts this ideal peer response automatically. It is not waiting for an actual tester. The original request can be functional, but the flow control goes to `7E0`, the physical request address of the ECU that sent the first frame.

The low-level `segment_payload()` helper only produces the single/first/consecutive frame bytes; it does not implement a transmitting protocol state machine. `isotp_trace()` adds the explanatory flow-control step. `IsoTpReceiver` is a genuine stateful parser for one already-filtered stream: it rejects bad sequence numbers, insufficient data, invalid lengths, optional padding mismatches, and simulated receive timeouts. Sequence numbers wrap after 15 to 0. Missing frames are rejected rather than silently reordered.

```
from obdii_lab.diagnostics import DiagnosticServer, IsoTpReceiver

server = DiagnosticServer(lambda: {"rpm": 2000, "coolant_c": 90})
reply = server.request("09 02")
receiver = IsoTpReceiver(timeout_s=1.0, padding_byte=0)
payload = None
for n, frame in enumerate(reply["frames"]):
    if frame["id"] == "7E8":
        payload = receiver.feed(frame["data"], timestamp=n * 0.001)
print(payload[3:].decode("ascii"))
# LAB00000000000001
```

The receiver does not generate flow control itself, and it must be fed one ECU’s data frames only. Production implementations need complete receive/transmit state machines, negotiated timing, addressing options, and transport error handling. The Linux ISO-TP stack provides an established path for a later virtual-CAN or CAN interface [R2].

5. Data scales and monitor flags

In the formulas below A and B are unsigned bytes; two-byte values are big-endian. Encoding clips to the representable interval and rounds to the closest representable value. The readings are model-derived; correct byte scaling does not make the plant a calibrated production engine.

PID	Decoding	Unit	Representable interval
04	100A / 255	% calculated load	0-100
05	A − 40	°C coolant	−40-215
0C	(256A+B) / 4	rpm	0-16,383.75
0D	A	km/h	0-255
0F	A − 40	°C intake air	−40-215
10	(256A+B) / 100	g/s air flow	0-655.35
11	100A / 255	% throttle	0-100

For Mode 01 PID 01, data byte A's high bit is MIL and its lower seven bits count confirmed emissions codes. This lab's byte B has comprehensive components supported (bit 2); bit 6 reports that simplified monitor incomplete until the core marks it complete. Other support bits and bytes C/D are zero. Therefore the response must never be interpreted as "this simulated vehicle has passed all emissions monitors." Unsupported monitors were never run.

6. Extension contracts

The public interface is intentionally independent of the physics implementation:

```
server = DiagnosticServer(sim.snapshot, sim.clear_emissions_dtcs)
result = server.request("01 0C", can_id=0x7DF)
# result: request_hex, response_hex (or None), meaning, frames, optional error
```

`snapshot()` supplies sensor/controller readings and dedicated emissions memory. `clear_emissions_dtcs()` clears simulated diagnostic memory while preserving the physical cause. The embedding application decides whether the ECU is powered and reachable; this diagnostic object itself remains available whenever called.

Reasonable next increments are:

1. Extend the optional python-can VirtualBus demonstration with scheduled delivery delays, dropped frames, and explicit bus-load estimates. Its existing ordered queues do not model arbitration or bitrate saturation.
2. Replace the ideal transcript with a full ISO-TP sender/receiver pair, including flow-control timeouts and block-size/STmin behavior.
3. Implement a gateway and additional ECU addresses from a documented target configuration.
4. Add validated module-specific DIDs/services and signal definitions. UDS service 0x22, for example, is a different diagnostic application, not an extra Mode 01 PID.

5. Add virtual CAN / SocketCAN as an explicit optional adapter; retain the deterministic in-process transport for repeatable tests.
6. Feed diagnostics from a real firmware execution environment only after its CPU/peripherals, sensor interfaces, scheduling, and network stack work. A `.bin` file is not a diagnostic-service implementation by itself.

No programming, security-access, immobilizer, or vehicle-flashing services are included. The current code is a local simulator with no hardware connector.

References and evidence boundaries

Accessed 9 November 2026. These are primary publisher/manufacturer sources. The standards' public scope pages are not substitutes for the complete normative documents.

- **[R1] SAE International, J1979_202505, E/E Diagnostic Test Modes.** Public scope and layer mapping. https://saemobilus.sae.org/standards/j1979_202505-e-e-diagnostic-test-modes
- **[R2] Linux Kernel documentation, ISO 15765-2 (ISO-TP).** Addressing, single/multi-frame transport, flow control, socket interface, and transport-error behavior. https://kernel.org/doc/html/latest/networking/iso15765-2.html
- **[R3] Elm Electronics, ELM327DSL datasheet.** Manufacturer's examples for OBD requests, RPM/coolant scaling, CAN addressing, DTC count format, and ISO-TP frame types; particularly printed pages 33-37 and 43-46. https://elmelectronics.com/wp-content/uploads/2020/05/ELM327DSL.pdf
- **[R4] SAE International, J1979_201202 public scope.** Identifies the J1979-DA and J2012-DA presentation-data references used for parameter and DTC definitions. Obtain the applicable normative editions when building a compliant implementation. https://saemobilus.sae.org/standards/j1979_201202-e-e-diagnostic-test-modes

The precise test vectors, Python implementation, fictional VIN, selected subset, and fault-demonstration timings are original lab choices. Toyota-specific supported PIDs, enhanced diagnostics, calibration identifiers, security/session requirements, and network maps have not been inferred from these generic references.

A path from this lab to a virtual vehicle and real firmware

1. The intended extension point

The useful separation is **plant → sensor interface → controller → actuator interface → plant**. Diagnostics observes controller state through a separate service boundary. A future virtual vehicle can provide road, driver and body dynamics while this lab supplies one or more ECU models. Alternatively, the lab's physical plant can stimulate a controller running in another process or emulator.

Version 1 is an executable model with deterministic simulation time and an implemented high-level JSON-lines process interface. It is not an instruction-set emulator, a hard-real-time system or a hardware-in-the-loop rack. The first integration should replace one controller while leaving the others as predictable models. Replacing the engine, battery and hybrid computers simultaneously makes a failed startup difficult to diagnose.

The interface you can run now

Start the bridge from the package directory:

```
python3 run.py --bridge
```

Send one JSON object per line on standard input. Each command produces one JSON reply on standard output. For example:

```
{"op":"reset","profile":"phev","config":{"initial_soc_pct":60}}
{"op":"step","dt_s":0.02,"inputs":{"ignition":true,"throttle":0.3}}
{"op":"observe"}
{"op":"diagnostic","payload":"01 0C","can_id":"7DF"}
{"op":"quit"}
```

`reset` creates a new manual simulation. `step` advances simulated time, with `dt_s` greater than zero and at most one second; repeat it to progress. `observe` returns the current state without advancing time. `diagnostic` executes the OBD payload handler. The response identifies the state schema as `obdii-lab.state.v1`; errors return `ok: false`. This bridge accepts driver/fault inputs and returns the complete high-level state. It does **not** expose ADC registers, raw actuator injection or an instruction-level ECU interface. It is a concrete starting boundary for a virtual-vehicle orchestrator, with richer controller replacement left to the next stages.

2. Four different meanings of “real firmware”

Route	What executes	What must be available	First useful milestone
Source-level soft ware-in-the-loop	Real controller logic compiled for the development computer	Source code, build dependencies, a defined I/O wrapper, rights to use it	Execute one periodic control function against recorded sensor inputs and compare outputs
Virtual ECU with software stack	Application plus selected scheduler, communication and diagnostic services	Buildable stack or a supplied vECU, configuration and OS/platform abstractions	Complete startup and a deterministic signal exchange with the plant
Binary emulation	Original target-machine instructions	Exact ISA, image layout, boot state and sufficiently accurate peripheral models	Reach startup, service a timer interrupt, then transmit one expected CAN message
Processor/hardw are in the loop	Code on a target processor or an actual ECU	Target hardware, controlled power and I/O, a test harness and time guarantees appropriate to the experiment	Reproduce one closed-loop function without attaching the controller to a drivable vehicle

MIL normally models controller and plant; SIL executes compiled controller code; PIL adds a target processor; HIL connects actual control hardware to a plant simulated in real time. These labels describe different test boundaries. A desktop Python program with a USB CAN adapter does not automatically become a complete HIL system. [S8]

3. Why a binary file alone is insufficient

A firmware image expects a particular computer. Before attempting execution, establish:

- **Identity and provenance:** ECU hardware part number, calibration/software IDs, original bytes, acquisition method, hash, image completeness and usage rights.
- **Processor:** exact ISA and variant, endianness, reset vectors, instruction mode, floating-point features and exception behavior.
- **Memory:** flash/RAM ranges, aliases, bootloader boundaries, calibration areas, nonvolatile data, executable permissions and initialization state.
- **Peripherals:** interrupt controller, clocks, timers, watchdogs, ADCs, PWM, DMA, CAN controllers/transceivers and any resolver/SPI/I²C devices the software accesses.
- **Startup environment:** power sequencing, reset causes, pin states, boot ROM services, checksums/signatures, nonvolatile configuration and expected peer modules.
- **Time:** instruction/peripheral synchronization, interrupt ordering, sampling deadlines and watchdog servicing.

A missing ADC-ready flag can trap firmware in a polling loop. An incorrect timer can change control periods. Returning zero for every unimplemented register can appear to “boot” while suppressing the behavior the experiment intended to study. Log unimplemented accesses and classify them rather than silently inventing device behavior.

QEMU’s own documentation explains that an image for one Arm machine will often fail on another even when the CPU family is supported. Renode provides facilities for register and peripheral models, but those

models still need to reproduce the relevant hardware contract. These are possible tools after target identification, not proof that an unidentified Toyota ECU is supported. [S6, S7]

The existing Prius firmware research reports static inspection of several Toyota-labelled engine files but leaves their MCU and complete memory layout unresolved; it also distinguishes those files from open replacement-controller firmware. Its documented Cortex-M example is a more tractable source-plus-binary exercise, but its CPU/address settings must not be copied to unrelated Toyota images. No original firmware is executed by this package. [U1]

4. A staged build plan with acceptance criteria

Stage	Add	Evidence required before progressing
A - Current learning model	Engine, battery, supervisor, longitudinal braking, fault scenarios, OBD subset	Repeatable runs, unit/sign checks, bounded state and fault-response tests
B - Explicit component interfaces	Separate sensor packets, actuator packets, timing/validity fields, reset and state serialization	Same scenario gives the same result after controller refactoring; stale/invalid signals are tested
C - External software controller	Host-compiled engine or BMU controller over a local IPC/shared-library boundary	Match a recorded reference trace within declared numerical tolerances; verify reset and missed deadlines
D - Vehicle co-simulation	External body/drivetrain, FMI wrapper or virtual network	Specify time ownership; show stable results when communication step is reduced; close the declared power balance
E - Target-specific vECU	ISA and peripheral models or a supplied virtual ECU	Boot and interrupt tests, peripheral contract tests, deterministic network traces, documented unsupported hardware
F - Bench HIL	Real ECU with appropriate electrical interfaces and a real-time plant	Characterized latency/jitter and failure response; reviewed pinout and power/I/O limits; controlled bench-only test procedures
G - Vehicle-specific validation	Measured maps, network definition and synchronized vehicle logs	Quantified error on independent data, parameter provenance, coverage of relevant operating conditions

The safest early firmware task is a small function with known inputs and outputs, such as a battery power-limit calculation. A large engine binary that expects immobilizer messages, an operating system and undocumented peripherals is a later research project. Authentication and protected flashing are not necessary for learning the control/plant boundary and are outside this package.

5. A proposed interface contract

The following richer packet is a **design proposal for the next version**, distinct from the implemented high-level bridge described above. Use named signals first; introduce byte packing at a separate adapter boundary.

```
{
  "schema": "obdii-lab-sensors/1",
  "simulation_time_ns": 1200000000,
  "sequence": 120,
  "step_ns": 10000000,
```

```
  "signals": {
    "engine_speed": {"value": 2000.0, "unit": "rpm", "valid": true, "age_ns": 0},
    "coolant_temperature": {"value": 90.0, "unit": "degC", "valid": true, "age_ns": 0},
    "battery_current": {"value": 60.0, "unit": "A", "valid": true, "age_ns": 20000000}
  }
}
```


An actuator response should include the input sequence it consumed, command timestamp, requested torque/power, validity and active limits. Commands must expire. Define reset behavior, retained state, rate limits, saturation and ownership of fault injection. Preserve physical state and measured state separately so the interface can represent sensor offset, quantization, noise, freeze, dropout and delayed delivery.

Use simulated time for reproducible offline tests. If an external controller has not responded at a synchronization point, either pause the simulation or apply the specified deadline policy; record which occurred. Wall-clock execution speed is a separate metric. Avoid silently advancing the plant with a command received from the future.

6. FMI, virtual CAN and data replay

FMI packages a model and its interface in an FMU. In **Co-Simulation**, a component normally brings its own integration or scheduling behavior; in **Model Exchange**, the importing environment performs integration; **Scheduled Execution** exposes partitions for an external scheduler. FMI 3 clocks can express timed controller activations. An FMU wrapper would be useful after the lab’s component interfaces and reset/state contracts are stable. It is not delivered in v1. [S4]

For diagnostics, a later Linux adapter can map the existing payload handler to a virtual CAN/ISO-TP endpoint. Linux’s ISO-TP sockets exchange diagnostic payloads while the kernel handles framing. Keep this transport adapter separate from the OBD application logic. Start on a virtual interface and test multiple responders, flow-control timing, request expiry and malformed frames before considering bench hardware. [S2]

A raw CAN replay is not a closed-loop vehicle: it repeats recorded messages regardless of new control requests. Replay is excellent for decoder and startup-sequence tests, but plant coupling needs commands to change subsequent sensor messages. Store log timestamps, bus/channel identity, arbitration IDs, data, direction and provenance. Distinguish replayed evidence from generated traffic in the trace.

7. What a future Toyota-specific integration still needs

Select an exact vehicle generation and hardware configuration before choosing gear relations, battery chemistry or signal definitions. Preserve the factory HEV architecture separately from a hypothetical plug-in conversion. Obtain synchronized measurements sufficient to identify engine and machine maps, battery electrical/thermal behavior, torque-request conventions, state transitions and network timing.

Model the Toyota planetary speed constraint explicitly if adopting the supplied Gen2 architecture. A single parallel engine/motor torque sum cannot reproduce the roles of MG1, MG2 and the power-split gearset. Keep kinematic constraints, power conversion and supervisory policy separate: knowing the gear teeth does not reveal the controller strategy. [T1, T3]

Finally, make validation claims narrow and measurable. “The ECU exchanged the expected startup frames” is different from “the torque controller is numerically equivalent,” which is different again from “the virtual vehicle matches measured driving behavior.” A successful simulator should show the evidence for the claim it makes and identify the components still represented by assumptions.

References and Google Drive source links are in **SOURCES.md**.

Existing bases: what to reuse and where

The most useful existing base depends on the layer being simulated. A transport library, a scan-tool adapter emulator and a physical vehicle model solve different problems. This lab keeps those boundaries separate so existing tools can be adopted without replacing the learning plant.

Selected first reuse: python-can VirtualBus

The optional `examples/python_can_demo.py` uses **python-can 4.6.1** as the CAN-message transport between a tester endpoint and an ECU endpoint in one Python process. The ECU uses this package's live `DiagnosticServer`, so response values come from the simulated engine. The example demonstrates single-frame requests, a segmented reply, flow control and payload reassembly. It is a real library integration, not a claim that the whole lab is based on python-can.

Install the optional dependency and run the demonstration from the package directory:

```
python3 -m pip install -r requirements-optional.txt
python3 examples/python_can_demo.py
```

The main dashboard, command-line scenarios and JSON bridge continue to run without this dependency. Installation needs an available Python package source; the dependency is not bundled into the archive.

The upstream VirtualBus connects endpoints on the same channel **within one process**. It preserves ordering and delivery but does not model bitrate, identifier arbitration or priority under bus load. This makes it useful for protocol integration tests, not a CAN timing benchmark. The demo selects only the virtual backend; it does not detect or open a hardware adapter. python-can's source license is GNU LGPL version 3. [S9]

Other useful bases and their boundaries

Existing project or artifact	Best use here	Current integration status	Boundary to preserve
python-can	Virtual message transport now; supported backend abstraction for later adapters	Optional executable example included	VirtualBus is not cross-process SocketCAN and does not simulate physical arbitration
Linux SocketCAN + can-utils	Later multi-process Linux virtual bus, capture, replay and kernel ISO-TP testing	Researched extension; not configured by this release	Requires the appropriate Linux interface/kernel setup; a replay trace does not respond to changing vehicle controls
Ircama ELM327-emulator	Later compatibility layer for software that expects ELM AT commands and a serial/TCP adapter	Researched candidate; not installed, vendored or coupled	Adapter/diagnostic emulation is separate from engine and battery physics; its license includes a noncommercial condition
Existing Prius Hybrid Lab in the user's Drive	Reference for a later Gen2 planetary and energy-flow plant	Its model notes were read; source was not imported	Prior 8.8 kWh PHEV is explicitly hypothetical; preserve its HEV/PHEV and evidence distinctions
QEMU / Renode	Target-specific firmware execution after hardware identification	Roadmap candidates	Need matching processor, memory, peripherals and startup environment

can-utils provides tools such as `candump`, `canplayer`, `cansend` and ISO-TP utilities. It is a suitable second transport step when separate processes need to communicate over Linux virtual CAN. The inspected `candump.c` and `isotpsend.c` headers offer GPL-2.0-only OR BSD-3-Clause; other files and dependencies need their own license review before redistribution. No can-utils binary or source is copied into this package. [S10]

ELM327-emulator already handles adapter commands and multiple simulated ECUs. Its Toyota preset is **Auris Hybrid**, selected with `scenario car`; its default preset is not a Prius. That dictionary can help test a scan application but does not establish factory behavior for another Toyota generation. The repository specifies **CC BY-NC-SA 4.0**, so its reuse terms differ from permissively licensed libraries. This release links to it as a candidate rather than incorporating its code or data. [S11]

A practical reuse order

Start by understanding the built-in model and running the optional python-can example. Next, use a Linux virtual bus if separate ECU processes are needed. Add an ELM-compatible facade when a specific scan application requires it. Introduce the existing Prius planetary model only after defining a common signal, time and energy contract; validate the resulting combined model with repeatable scenarios. Treat source-level controller integration as a separate milestone from executing an original ECU binary.

The existing Drive model notes give a useful starting interface and explicitly list assumptions. They report a 201.6 V HEV baseline and a hypothetical 8.8 kWh extension, with simplified efficiencies and supervisory logic. This is a candidate physical-model base, not a factory calibration dataset or a firmware emulator. [U2]

Source links and evidence scope are in **SOURCES.md**; binary execution requirements are in **FIRMWARE_ROADMAP.md**. Each optional dependency should be pinned and its own tests run when introduced, so a future update does not silently change the simulator's behavior.

EngDoc: a first interactive coolant-sensor worksheet

OBD-II Learning Lab · Version 0.3.0 · 9 November 2026

What is ready

The supplied EngDoc source has been inspected, and the lab now includes **examples/engdoc_coolant_sensor.json**, a native `engdoc-native-draft/0.3` worksheet. It brings a circuit explanation, an embedded diagram, adjustable inputs, calculated results, a small table and a line plot into one document. The file follows the actual uploaded native contract and its publishing example. Native contract · Publishing example

The question is deliberately small: **What reaches the engine computer when the coolant gets warmer, and what changes if the sensor connection is broken?** This is a useful first connection between a readable engineering document and the larger virtual vehicle. It does not require entering hexadecimal codes or writing a program.

The worksheet recalculates a steady electrical circuit when an input changes. The browser Learning Lab adds simulated time, changing physical temperature, computer decisions, fault timers and warning memory. The two tools explain different parts of the same measurement path.

Place to learn	What you can change	What you can observe
EngDoc coolant worksheet	Actual temperature, connection state, reference supply, fixed resistor	Sensor resistance, computer-side voltage, branch current and a temperature curve
Browser learning book	Driver requests, starting conditions and a named fault	Physical response, measured information, controller decisions, timing and diagnostic history
PDF handbook	Read the same explanations and exercises	Reference pictures and worked reasoning; the PDF itself does not run the model

Open and try the worksheet

Use the supplied EngDoc application’s native-document import or open workflow to select **examples/engdoc_coolant_sensor.json**. It requires the `/0.3` native draft capability because the document contains an embedded PNG. The exact menu label depends on the EngDoc build. A complete EngDoc application import and rendering session has not yet been run for this deliverable; the verification below covers the actual native validator and the calculation data. See the supplied EngDoc README for that application’s startup instructions.

Start with these settings:

Control	Initial value	Plain meaning
Actual coolant temperature	25 degrees Celsius	The physical condition at the sensor

Control	Initial value	Plain meaning
Sensor connection	0	Connected; choose 1 to disconnect
Computer reference supply	5 volts	The supply used by this measurement circuit
Fixed resistor inside computer	2,200 ohms	One of the two parts that divide the voltage

The connection control uses the choices **0 = connected** and **1 = disconnected** because the native contract stores scalar choices. The worksheet repeats that meaning beside the control and results. These numbers are switch positions, not diagnostic codes. The browser book provides named fault buttons for the same learning idea.

Try the three experiments in order:

1. **Warm the connected sensor.** Keep the connection at 0. Change temperature from 25 to 90 degrees Celsius. The signal falls from about 2.660 to 0.610 volts. Warmer coolant lowers the sensor resistance, which lowers the measured junction voltage.
2. **Break the information path.** Return to 25 degrees Celsius, then select connection 1. The signal rises to 5.000 volts and sensor-branch current becomes zero. Move temperature again: the intrinsic sensor resistance changes, but the computer-side signal stays at the reference supply. The broken path hides the temperature change.
3. **Repair the connection.** Select 0 again. The signal returns to the value associated with the current physical temperature. This worksheet has no warning memory. In the browser lab, repair and clearing remembered warnings are separate actions.

The six-row table uses fixed temperatures from −20 to 110 degrees Celsius. It follows your selected supply, fixed resistor and connection state, so it shows the wider relationship alongside the single-temperature control. The line chart joins those six points as a visual guide. The embedded circuit picture stays still; it is a reference drawing, not an animated switch or a vehicle connector pinout.

The small calculation underneath

The illustrative sensor is an NTC thermistor: its resistance decreases as it gets warmer. The worksheet uses the same illustrative beta approximation and default component values as the lab's `ect_circuit()` function in **obdii_lab/core.py**. These are teaching parameters, not extracted Toyota calibration values.

The equations are:

- Temperature for the calculation: `T_K = T_C + 273.15`.
- Sensor material resistance: `R = 2500 × exp(3500 × (1 / T_K − 1 / 298.15))` ohms.
- Connected signal voltage: `V_connected = V_reference × R / (R_pullup + R)`.
- Selected signal voltage: `V_signal = (1 − open) × V_connected + open × V_reference`.
- Branch current: `I_mA = (1 − open) × 1000 × V_reference / (R_pullup + R)`.

Here, `open` is restricted by its selector to 0 or 1. The displayed resistance is the **intrinsic sensor resistance if connected**. The disconnected branch itself is open; it is not assigned that finite resistance. No

inverse-temperature formula is applied to an open signal, so the worksheet does not manufacture a plausible temperature from a broken measurement path.

The source adapter's function allowlist includes `exp()`, and it permits the simple arithmetic and bound symbols used here. Values use the `float64` profile with units written in their labels. This avoids mixing absolute-temperature units with an exponential argument. It also means the worksheet does not perform unit conversion or dimensional checking: enter the units shown by each control. Uploaded expression adapter

The circuit assumes an ideal reference supply and return, no wire resistance, no input leakage and no measurement noise. It has no analogue-to-digital quantization. The browser lab's reported voltage includes its selected ADC rounding, so it can differ slightly from the worksheet's ideal junction voltage. Compare the worksheet with the lab's ideal circuit node when checking exact values.

What was verified

The JSON passed the **actual uploaded** `validateNativeDocument()` function after its TypeScript source and local dependencies were transpiled. This checked the `/0.3` schema, ownership of inputs and calculations, value references, table and chart references, embedded PNG structure and size limits. The document has 37 nodes and 29 values and is about 175 KiB, including its own diagram.

Every calculation expression was also evaluated by an independent, restricted arithmetic evaluator. Results were compared with the lab circuit for 54 combinations of temperature, supply voltage, fixed resistor and connection state, including the allowed input boundaries. All comparisons passed. The six table rows were checked separately against the same circuit function.

Healthy case, default circuit	Signal voltage	Sensor-branch current
−20 degrees Celsius	4.507752 V	0.223749 mA
0 degrees Celsius	3.844623 V	0.525172 mA
25 degrees Celsius	2.659574 V	1.063830 mA
60 degrees Celsius	1.243604 V	1.707453 mA
90 degrees Celsius	0.610156 V	1.995383 mA
110 degrees Celsius	0.387639 V	2.096528 mA

At 25 degrees Celsius the sensor resistance is exactly 2,500 ohms in this model. With the connection open, every temperature case produces the selected reference voltage and zero branch current. These are useful known answers when first opening the worksheet.

Verification boundary: schema validation, numerical agreement and inspection of the expression adapter are complete. The full EngDoc expression runtime, editor rendering, interactive input widgets and export workflow have not been exercised here. A native document passing its contract does not by itself prove those application paths. The worksheet contains no document-supplied code and does not execute vehicle firmware.

How this grows toward the user’s workbench

The next useful step is to open this one worksheet in the supplied EngDoc build and verify its controls, table, chart and printed layout there. After that, the same pattern can be reused for a wheel-speed measurement or a battery-current measurement. A later bridge could pass explicitly named, unit-labelled values between an EngDoc worksheet and the running lab. Such a bridge needs defined ownership, update timing, validity flags and reset behavior before it can represent controller interactions reliably.

The native worksheet is currently a separate, bounded document. No live EngDoc-to-simulator synchronization is implemented. Original firmware still needs a specified processor, memory and peripheral environment; neither this JSON nor the browser teaching controller supplies that environment. The firmware roadmap explains those later boundaries.

The supplied EngDoc project was used as the format reference. The separately named Wordcraft upload folder has not been evaluated as a running application for this release, and no Wordcraft integration is claimed. This first worksheet can be reviewed independently before choosing that integration boundary.

Sources and evidence boundaries

Version 0.3.0 · Research date: 9 November 2026. The documentation paraphrases the sources; original documents remain in their existing locations. The simulator source code and tests are the authority for implemented behavior, not the source list below.

Public primary sources

ID	Source	What it supports
S1	California Air Resources Board, On-Board Diagnostic II Systems Fact Sheet, 19 September 2019	OBD-II monitors emissions-related components and stores malfunction information. The document's regulatory dates are jurisdiction-specific and are not used as a global applicability claim.
S2	Linux kernel documentation, ISO 15765-2 / ISO-TP	Physical versus functional addressing, single/multi-frame transport, flow control, sequence handling and the Linux socket interface. It does not provide a Toyota message database.
S3	Toyota Motor Corporation, Toyota Develops New Hybrid System, 17 April 2003	Manufacturer overview of THS-II power splitting, propulsion modes and regenerative braking. Describes that historical system, not every later Toyota hybrid.
S4	Modelica Association, Functional Mock-up Interface specification 3.0.2, 27 November 2024	FMU packaging, Model Exchange, Co-Simulation, Scheduled Execution and clocks. FMI export is a proposed extension, not a delivered capability.
S5	Bosch Mobility, Electronic stability program	Functional roles of wheel speed, steering and yaw measurements; propulsion reduction and individual-wheel braking. The lab does not implement Bosch software or calibration.
S6	QEMU project, Arm System emulator	A matching CPU architecture alone is insufficient; firmware depends on a supported board/machine and its devices. No assertion is made that QEMU supports an unidentified Toyota ECU.
S7	Renode project, Peripheral modeling guide	Register/peripheral models and the relationship between CPU accesses, memory and the simulated system bus. No Renode peripheral implementation is included in this release.
S8	MathWorks, What Is Hardware-in-the-Loop? and SIL, PIL and HIL tests	Terminology for model, compiled software, target processor and controller-hardware testing; equivalence testing. MathWorks products are not required by this lab.
S9	python-can project, Virtual interface documentation, source repository, license	Same-process VirtualBus behavior and limitations; optional transport base. Stable documentation inspected identifies version 4.6.1; repository license is GNU LGPL version 3.
S10	Linux-CAN, can-utils repository, candump source, isotpsend source	Capture/replay and ISO-TP utilities for a later Linux virtual-CAN workflow. Those two source headers specify GPL-2.0-only OR BSD-3-Clause; inspect individual files rather than assuming a uniform repository license.
S11	Ircama, ELM327-emulator repository, license	Adapter/scan-client emulation and an optional Toyota Auris Hybrid request dictionary. It is not a Prius firmware/physical model. The repository identifies CC BY-NC-SA 4.0.

ID	Source	What it supports
S12	Texas Instruments, Monitoring NTC Thermistor Circuit With Single-Ended ADC, SBAA338A, revised June 2022	NTC divider topology, beta resistance/temperature relationship and ratiometric ADC principle. The lab's 5 V, 2,200 Ω, 2,500 Ω, 3,500 K and timing choices are synthetic; this source does not establish a Toyota sensor calibration or pin assignment. ADC settling and tolerances in the application note are not implemented here.
S13	Elm Electronics, ELM327 data sheet, version 2.3	Printed p82, Figure 8: vehicle J1962 connector numbering and orientation. Printed p84, Figure 9: adapter reference circuit and protocol pin roles. Supports the DLC reference, not execution of an ELM adapter or analog CAN waveforms.
S14	ScanTool.net / OBD Solutions, Which OBD-II protocol is supported by my vehicle?	Vehicle-connector diagrams, ground/supply functions and protocol-dependent pin assignments. The CAN lesson uses pins 6/14; legacy pins are reference-only. Historical market/date statements are not used as current global applicability guidance.

Toyota documents read from the user's Google Drive

These titles and file identities were verified through Google Drive, and their readable content was inspected. The local package links to them rather than redistributing their full contents. Page references below are printed document pages.

ID	Document	Specific evidence used
T1	Toyota_Course_071_Hybrid_System_Operation.pdf	Page 2-2: control-system components; 2-7 to 2-8: 30/78 tooth counts and engine/carrier, MG1/sun, MG2/ring connections; 2-12 to 2-15: inverter, converters and HV ECU roles.
T2	Toyota_Course_071_High_Voltage_Battery.pdf	Page 3-3: 28 modules and 201.6 V Gen2 pack; 3-4 to 3-5: battery monitoring; 3-6: resistor-assisted SMR energizing sequence.
T3	Toyota_2004_Prius_THSII_Features.pdf	TH-4 to TH-6: planetary power splitting, operating modes and regenerative-brake cooperation.

These documents are historical training/reference material. This lab does not reproduce servicing procedures or establish present-day repair instructions for any vehicle. Exact vehicle generation, market, hardware part numbers and calibration IDs would be required for a specific implementation.

Existing user research read for continuity

ID	Document	How it was used
U1	Prius_firmware_and_Ghidra_guide.html, 2 November 2026	Prior research separates Toyota-labelled engine calibration images, absent authenticated HV/battery images, and open replacement firmware. Its stated uncertainties about MCU identity, memory mapping and compatibility are retained. This turn did not re-download or independently execute the binaries.
U2	MODEL_NOTES.txt	Prior quasi-static Prius model explicitly labels controller choices and the 8.8 kWh PHEV extension as hypothetical. It informs the separation between manufacturer specifications and illustrative strategy; its JavaScript model was not imported into this package.

Evidence labels used in this package

Manufacturer fact means a scoped statement supported by a Toyota source, such as a historical gear connection or nominal battery voltage. **Standard/protocol reference** means a diagnostic or interoperability concept supported by its identified specification or official implementation documentation. **Teaching assumption** means a parameter, threshold, signal, control law or scenario chosen for learning. **Verified implementation result** means behavior observed by running this package; it does not imply measured vehicle fidelity.

All generic hybrid parameters, control periods, fault debounces, synthetic network signals, torque policies and simplified tire/thermal behavior belong to the teaching-assumption category unless an individual item says otherwise. No Toyota CAN identifier, proprietary PID, DTC subcode, firmware compatibility or production protection threshold is inferred from a plausible-looking number.

The new connector reference applies the same boundary to schematics: the DLC’s referenced positions and roles are distinct from fictional J-ECT and X1 lesson assignments. X1.1 is an internal reference test point, not a claimed physical ECU cavity. No real Toyota connector part number or harness layout is inferred. The 5 V ECT circuit and ideal 12-bit conversion are executable teaching models; their computed values are not recorded vehicle measurements. The workspace guide explains how invalid, substituted and stale values remain visible with their validity context.

For a later vehicle-specific model, maintain a provenance record per parameter: symbol, unit, value or map, vehicle generation, source page or dataset, measurement conditions, uncertainty, software revision and license. A publicly visible file can still have redistribution restrictions. Reference source locations and preserve notices when including third-party firmware or model code.