



To provide downstream mobile and desktop frontends with a clean, unified view of vehicle operations, the VCI core translates heterogeneous low-level automotive registers into predictable application-layer system variables (Technical ... p. 7, driver/obd\_driver.c):

- **Ignition Status (Terminal 15 Validation):**

The kernel module actively traces pin voltages on the physical OBD-II connector interface (driver/power\_mgmt.c). This raw hardware signal is captured bitwise and mapped directly onto the VCI\_STAT\_IGNITION\_ON mask inside the shared memory region (src/core/main.c). The serialization engine converts this flag into a simple boolean value ("ignition": true), allowing the Android app to automatically lock or unlock interactive interface screens depending on whether the vehicle is on contact (src/gateway, web/app.js).

- **Bus Error Modes & Bus-Off Interrupt Tracking:**

When the hardware CAN controller (MCP2515) catches transmission faults exceeding strict ISO 11898-2 thresholds (such as loose wiring or termination errors), it flips into an automatic *Bus-Off* recovery loop (Technical ... p. 9, driver/obd\_driver.c). The driver context logs these anomalies instantly via the VCI\_STAT\_BUS\_OFF registry (driver/obd\_driver.c). The API layer exposes these states via the global error tracking key ("errors"), enabling real-time monitoring of connection stability (src/gateway, web/app.js).

- **Diagnostic Session Tracking:**

When transitioning from standard data polling (OBD Mode 01) to extended engineering sessions (UDS Service 0x10 subfunction 0x03), the internal state transition is logged inside the active core framework (wwh\_obd\_stack.c, uds\_iso14229.c). The system locks out volatile telemetry requests during active security access negotiation loops, preventing bus congestion while firmware or fault clearing commands are executing (Technical ... pp. 4, 8).

---

## Encoding Efficiency Constraints

In resource-constrained embedded environments like the Raspberry Pi, dynamic string manipulation libraries (such as malloc allocations or automated heap growth models) introduce a high risk of memory fragmentation, buffer leaks, and non-deterministic timing jitter (Technical ... p. 3). To maintain strict real-time compliance during high-frequency telemetry encoding, the serialization module (json\_formatter.c) enforces a fixed-memory allocation paradigm (Technical ... p. 3, json\_formatter.c).

The serialization engine operates under three core structural constraints:

- **Pre-Allocated Static Target Buffers:** All string generation loops take place inside a pre-bound target memory array (JSON\_BUF\_SIZE = 4096U), entirely bypassing runtime heap allocation commands (kmalloc/malloc) (Technical ... p. 3, src/gateway).
- **Defensive Index Point Guarding:** Text rendering is managed via strict memory-offset calculations driven by boundary-checked sprintf routines (json\_formatter.c, src/gateway). The remaining buffer length (max\_len - offset) is re-validated at every processing point; if string truncation or array exhaustion is detected, execution halts safely before a buffer overflow occurs (json\_formatter.c).

- **Low-Overhead Number Formats:** Floating-point tracking entries (such as engine coolant temperature or common-mode battery voltage) are parsed using lightweight precision definitions (`%.2f`), keeping the final string output tightly packed and optimized for immediate parsing by the mobile client application (`src/analysis/battery_health.c`, `json_formatter.c`).

## 2.g.2 Core JSON Telemetry Schema Spec (`/api/telemetry`)

### Structural Composition

The real-time telemetry stream distributed via the embedded endpoint (`/api/telemetry`) is structured as a compact, single-tier flat array wrapper designed for efficient, low-overhead parsing on the client side (`json_formatter.c`, `src/gateway`). This payload architecture is tailored specifically to meet the high-speed data ingestion requirements of the native Kotlin-based Android application (Libre Diag... p. 5, `web/app.js`).

By encapsulating individual engineering parameters inside an outer root array object named `"telemetry"`, the schema avoids nested tree navigation (`json_formatter.c`). This configuration minimizes the CPU memory footprint during deserialization blocks, which is ideal for performance-sensitive mobile devices (Technical ... p. 4, `web/app.js`).

json

```
{
  "telemetry": [
    {
      "id": "0x0C",
      "name": "Engine RPM",
      "val": 2400.00,
      "unit": "rpm"
    },
    {
      "id": "0x0D",
      "name": "Vehicle Speed",
      "val": 90.00,
      "unit": "km/h"
    }
  ],
  "ts": 1482390
}
```

---

### Field-Level Definition Ledger

Each discrete element nested within the telemetry array contains four standardized, tightly controlled key-value pairs that translate raw bitstreams into validated metrics (`json_formatter.c`):

- **id (String):**  
A 4-character alphanumeric string representing the standardized OBD-II Parameter ID (PID) or UDS Data Identifier (DID) in hexadecimal format (e.g., `"0x0C"`, `"0x0D"`, `"0x05"`) (`json_formatter.c`). The Android client utilizes this key as a strict unique identifier to map incoming variables directly to the corresponding UI dashboard gauges, bypassing slow string-matching procedures (`web/app.js`).
- **name (String):**  
The alphanumeric label string derived directly from the master `diagnostic_db.csv`

mapping configuration (json\_formatter.c). This value represents the human-readable description of the vehicular sensor (e.g., "Engine RPM", "Vehicle Speed", "Coolant Temp") (json\_formatter.c).

- **val (Float):**

The real-time, double-precision floating-point engineering value (json\_formatter.c). This metric is calculated deterministically within the hardware abstraction layer by processing raw hex bytes through fixed-point mathematical formulas (e.g., converting bytes A and B into scaled engine speed) (obd\_formulas.c).

- **unit (String):**

A strict, standardized metrics symbol string denoting the scientific unit of measurement assigned to the processed value (e.g., "rpm", "km/h", "°C", "g/s", "%")

## 2.g.3 System VCI Status Schema Spec (/api/status)

### Hardware Telemetry Registry

The system state monitoring pipeline utilizes a specialized, lightweight JSON data schema served via the /api/status endpoint (src/gateway). While the telemetry endpoint focuses exclusively on high-frequency vehicular ECU data streams (src/gateway), the VCI Status schema functions as an embedded hardware registry (src/gateway). It is designed to capture the structural health, raw electrical conditions, and network performance boundaries of the physical Raspberry Pi VCI node and its custom diagnostics shield in real-time (Libre Diag... p. 4, src/gateway).

This schema provides the native Android application with instant infrastructure diagnostic metrics (Libre Diag... p. 5, web/app.js). By evaluating these environmental baselines, the frontend client can perform safety cross-checks before initializing intrusive UDS routine executions or firmware flash sessions (Technical ... p. 8, src/gateway).

json

```
{
  "voltage": 14.20,
  "bus_load": 12,
  "errors": 0,
  "ignition": "true"
}
```

---

### Parameter Breakdown Ledger

Every variable encapsulated within the System VCI Status schema corresponds directly to an active tracking ledger register mapped inside the core memory boundary:

#### 1. voltage (Floating-Point)

- **Technical Definition:** Delivers a precision floating-point metric representing the common-mode DC electrical potential tracked at Pin 16 of the standard OBD-II connector matrix (src/analysis/battery\_health.c, src/gateway).
- **Hardware Mapping:** This value directly monitors the vehicle's permanent battery supply line (**Terminal 30**) (src/analysis/battery\_health.c).
- **Operational Significance:** It feeds the user-space battery health monitor to flag critical low-voltage drops or charging system alternator diode failures (src/analysis/battery\_health.c). Any

reading below 11.8V automatically restricts high-current diagnostic operations (src/analysis/battery\_health.c, src/gateway).

## 2. bus\_load (Unsigned Integer)

- **Technical Definition:** Documents the estimated bandwidth utilization and controller queue saturation profile of the network layer, represented as a strict percentage integer (0 to 100) (src/gateway, web/app.js).
- **Hardware Mapping:** Calculated by evaluating the ratio of successfully parsed Layer 2 CAN frames against the total physical transmission limits of the MCP2515 transceiver chip across a rolling 100ms window (src/analysis/health\_monitor.c, src/gateway).
- **Operational Significance:** Serves as an early warning indicator for network congestion or bus flooding events (src/analysis/health\_monitor.c). A threshold value exceeding 90% immediately triggers traffic dampening to prevent unintended ECU timeouts (src/analysis/health\_monitor.c).

## 3. errors (Unsigned Integer)

- **Technical Definition:** A global, non-volatile diagnostic accumulation register logging the total count of software, hardware, or transmission anomalies caught during the active execution cycle (src/gateway, web/app.js).
- **Hardware Mapping:** Pulls error flags systematically generated across both the kernel driver (e.g., SocketCAN frame drops, SPI bit-timing misalignments) and user-space layers (e.g., failed TLS handshakes, JSON formatting bounds overrides) (Technical ... p. 11, driver/obd\_driver.c, src/gateway).
- **Operational Significance:** Provides the client application with an automated tracking tool to gauge overall VCI link stability. Any non-zero increment forces the front-end display to flag systemic errors to the user.

## 4. ignition (String Enclosed Boolean Variant)

- **Technical Definition:** A clear state-tracking token indicating the exact physical state of the vehicle's switched ignition circuit (**Terminal 15**) (src/gateway, web/app.js).
- **Hardware Mapping:** Maps to a hardware interlock circuit on the custom VCI diagnostics shield, tracking high/low input transitions on a dedicated GPIO interface line (driver/power\_mgmt.c, src/gateway).
- **Operational Significance:** It evaluates to "true" or "false" to safely gate dangerous bidirectional diagnostic interactions (Technical ... p. 8, src/gateway). For example, a UDS Mode 04 fault-clearing request requires confirmation that ignition is "true" while Engine Speed is 0 RPM to prevent accidental memory resets while the vehicle is in motion (Technical ... p. 8).

### 2.g.4 UDS Diagnostic Trouble Code (DTC) Report Schema Spec Multi-Frame Assembler Layout

Unlike standard, single-frame real-time telemetry pipelines, Diagnostic Trouble Codes (DTCs) frequently generate payloads that exceed the 8-byte limit of standard CAN 2.0B frames (Technical ... p. 5). The DTC report schema maps the data blocks assembled by the **ISO 15765-2 (ISO-TP)** transport layer into structured diagnostic arrays (Technical ... p. 5).

When a diagnostic scan is triggered (via UDS Service 0x19 or OBD Mode 03), the kernel-assisted transport stack coordinates the network handshakes (Technical ... p. 5). It receives a *First Frame (FF)* from the target ECU, automatically replies with a *Flow Control (FC)* frame, and collects all subsequent *Consecutive Frames (CF)* (Technical ... pp. 3, 5). Once reassembly is completed, the continuous, validated hex buffer is parsed by the user-space engine and mapped into a memory-safe JSON array object (Technical ... p. 5, json\_formatter.c).

json

```
{
  "dtc_report": {
    "count": 2,
    "codes": [
      {
        "code": "P0301-1C",
        "severity": "CLASS_B",
        "description": "Cylinder 1 Misfire Detected"
      },
      {
        "code": "U0100-00",
        "severity": "CLASS_A",
        "description": "Lost Communication With ECM/PCM"
      }
    ],
    "mil_status": 1,
    "timestamp": 1712784321
  }
}
```

---

## Alphanumeric Code Mapping Structure

The `dtc_report_t` core structure converts raw binary diagnostic responses into structured JSON keys according to strict **SAE J1979 / ISO 15031-6** specifications (Technical ... pp. 5, 7):

- **count (Unsigned Integer):**

Represents the absolute number of confirmed fault codes found in the vehicle's diagnostic memory bank (Technical ... pp. 5, 7). It defines the total data array bounds, allowing the front-end application to size its scrollable overlay lists dynamically without memory fragmentation (Technical ... p. 7).

- **codes (Array of Structured Objects):**

A static array containing the alphanumeric trouble strings (Technical ... p. 7). Each entry is processed through bitmasking logic (Technical ... p. 5). The upper bits of Byte A ([7:6]) define the subsystem category prefix (00=P, 01=C, 10=B, 11=U), while bits [5:4] define the second digit (0=Generic, 1=Manufacturer Specific) (Technical ... p. 6). The remaining hex nibbles are parsed using `snprintf` to output standard strings (e.g., "P0301") (Technical ... pp. 6-7, `wwh_obd_stack.c`).

- **mil\_status (Unsigned Integer):**

An explicit flag verifying the functional status of the dashboard Malfunction Indicator Lamp



```

| [ Start 5Hz Polling Engine ] |
| --- 5. GET /api/telemetry ----> |
| <== 7. Stream JSON Payload ==== | <-- 6. Zero-Copy RAM Parse -- |
| --- 8. GET /api/status -----> |
| <== 9. Stream Hardware JSON === |

```

1. **Layer 4 Socket Initialization:** The client opens an outbound TCP channel directed at the host's active gateway network interfaces on either the secure port (8888) or standard port (80) (src/gateway).
2. **Cryptographic Handshake Verification:** Connections on port 8888 are immediately enveloped inside an OpenSSL runtime context (src/gateway). The framework enforces a 1-Round Trip Time (1-RTT) **TLS 1.3 handshake**, exchanging ephemeral Diffie-Hellman parameters to insulate subsequent packets from sniffing or interception (src/gateway).
3. **Application-Layer Key Interlock:** Once the encrypted channel is established, the client dispatches its internal validation payload string (AUTH\_TOKEN\_EXPECTED) (src/gateway). The daemon executes an array-comparison filter; a match unlocks the shared data access layer, while an invalid token triggers an immediate session reset (src/gateway).
4. **High-Priority Asynchronous Polling Loop:** Upon authorization, the client's internal loop engine engages an asynchronous polling frequency locked to a **200ms interval (POLL\_INTERVAL = 200)** (web/app.js).
5. **Parallel Resource Dispatching:** The client utilizes concurrent execution structures (Promise.all()) to fire simultaneous, non-blocking requests to the /api/telemetry and /api/status endpoints (web/app.js). The VCI user-space daemon catches these asynchronous triggers via select(), extracts the data from the memory-mapped vci\_shared\_mem\_t cache, serializes the numbers using boundary-safe sprintf routines, and returns compact JSON blocks back to the client at a **5Hz refresh profile** (lib/remote\_vci\_bridge.c, json\_formatter.c, src/gateway, web/app.js).

### State Mutation Actions (/api/clear\_dtc)

Unlike passive tracking endpoints that only read sensors, executing a state-mutation action—such as an active clearing of diagnostic trouble codes—requires strict safety mechanisms (Technical ... p. 8, web/app.js). Purging non-volatile memory registers via **UDS Service 0x14 (or OBD Mode 04)** could cause systemic vehicle faults or module damage if triggered while the automobile is operating (Libre Diag... p. 6, Technical ... p. 8).

Therefore, the API defines a rigorous multi-layered execution path combining physical, front-end, and back-end structural safeguards (Technical ... p. 8, web/app.js):

#### 1. Front-End Human Verification Layer (Hold-to-Confirm)

- **Mechanism:** The user-space client console or Android application suppresses instant click execution on the clear interface button (Technical ... p. 8, web/app.js).

- **Execution Boundary:** The front-end framework listens for a prolonged `mousedown` trigger, launching an internal tracking sequence loop (`web/app.js`). The operator must continuously maintain button contact for **3000 milliseconds** (`HOLD_TIME_THRESHOLD = 3.0f`) (Technical ... p. 8, `web/app.js`).
- **Anti-Bounce Protection:** If the pointer slips outside the interactive bounding box parameters due to garage vibration, an immediate reset loop clears the timer buffer, preventing accidental command execution (Technical ... p. 8, `web/app.js`).

## 2. Transmission Payload Routing

- **Mechanism:** Once the 3-second hold window closes successfully, the front-end transforms the interaction into an explicit **HTTP POST request** targeted at the `/api/clear_dtc` endpoint (`web/app.js`).

```
+-----+
| HTTP POST /api/clear_dtc HTTP/1.1          |
| Host: 192.168.1.1                          |
| Content-Type: application/json              |
| Content-Length: 35                          |
| {"confirm_clear": true, "sid": "0x14"}      |
+-----+
```

## 3. Back-End Middleware & Physical Interlock Verification

- **Mechanism:** Upon ingestion of the POST payload, the core C daemon routes the command through a strict physical verification filter before wrapping the diagnostic bits for the CAN layer (Technical ... p. 8, `src/gateway`).
- **Safety Rules Evaluation:** The backend cross-checks the shared memory registry status (`vci_shared_mem_t`) to verify that two parameters are satisfied: **Engine RPM must be exactly 0** (ensuring the vehicle is stationary) and **Battery Voltage must be >12.0V** (ensuring the Terminal 15 ignition line is on, providing adequate power to rewrite the flash memory configuration banks) (Technical ... p. 8, `src/analysis/battery_health.c`).
- **Hardware Execution:** If these conditions are verified, the daemon passes the command to the ISO-TP network stack, broadcasting a 4-byte payload [`0x14`, `0xFF`, `0xFF`, `0xFF`] to the ECU address group to clear fault memory (Technical ... p. 8, `uds_iso14229.c`). Following confirmation from the vehicle, the shared registers (`shm->error_count` and `shm->mil_active`) are safely set to `0U` (Technical ... p. 8).