1. Diagnostic Service Mode 03 Architecture

1.1. SAE J1979 Emission-Related Fault Register Topologies

The architectural implementation of **Service 03** within the application platform complies fully with the **SAE J1979** and **ISO 15031-5** regulatory frameworks. This diagnostic service mode is explicitly dedicated to the extraction of confirmed emission-related and powertrain-critical Diagnostic Trouble Codes (DTCs). Unlike real-time operational parameters under Service 01 which transmit continuous data waves, Service 03 queries a static non-volatile onboard fault memory buffer. This partition is updated dynamically by the vehicle's diagnostic executive only when an anomaly is verified across consecutive driving cycles.

1.2. Service 03 Multi-ECU Response Mapping

When the application dispatches a raw `"03"` serial query string, the command is transmitted as a functional broadcast frame targeted at the standard network identifier (`0x7DF`). In a modern automotive topology containing multiple control modules (e.g., Engine Control Module and Transmission Control Module sharing the same link), the system is built to ingest concurrent asynchronous message frames from discrete physical node addresses. The processing architecture separates these incoming data blocks by network origin, channeling them to the validation pipelines without allowing data packet overlap or cross-module buffer corruption.

1.3. Frame Structures and Positive Acknowledgment Verification

A successful fault memory extraction sequence triggers a standard positive response message framework, denoted by adding a `0x40` offset to the request identifier, which results in the positive reflection byte **`43`**. The structure of a standard Service 03 CAN message frame comprises:

- **Byte 0:** Data Length Code (DLC) indicating subsequent byte validation parameters.
- **Byte 1:** Service Identification reflection token (`43`).
- **Bytes 2 to 7:** Concatenated 2-byte hexadecimal payload slots holding active trouble codes. The application’s data link layers actively scan the head of the sanitized incoming buffer string for this reflection signature `43`, creating an authorization gate that blocks processing if the frame contains unexpected formatting noise or protocol rejections.

---

2. Low-Level Bitstream Decoupling & Alphanumeric Translation

2.1. The 2-Byte (16-bit) DTC Mapping Matrix

Each independent Diagnostic Trouble Code is represented within the automotive microcontrollers as a compressed **2-byte (16-bit) boundary matrix**. This compact structure is defined by the **SAE J1979** and **ISO 15031-6** standards to maximize information density over standard CAN architectures. The raw data payload arrives at the data management layer as a

set of four alphanumeric hexadecimal characters (e.g., `0103`), which encapsulates both the physical system boundary where the fault originated and the precise technical failure code.

2.2. Bit-Level Masking for Prefix Character Extraction (P, C, B, U)

To transform these raw byte fields into a standard five-character alphanumeric code, the `DtcDecoder` execution matrix applies accurate **bitmask operations** over the highest order nibble (the first 4 bits of the primary byte). The two highest bits (Bit 7 and Bit 6) dictate the alpha prefix identifying the core automotive system domain:

- `00` → **P** (Powertrain - Engine, Transmission, and Emission components)
- `01` → **C** (Chassis - Steering, Traction, and Anti-lock Braking systems)
- `10` → **B** (Body - Passenger safety, Climate Control, and Interior electronics)
- `11` → **U** (Network - Inter-ECU serial communication links and CAN Bus anomalies)

The algorithm masks these bits using a bitwise AND operator against the binary bitmask `0xC` (`1100`) and executes a logical right-shift operation (`>> 2`) to produce a zero-offset index pointing directly to the alphanumeric lookup array.

2.3. System Classification and Code Type Identification (Bits 5-4 Validation)

The subsequent two bits within the high nibble (Bit 5 and Bit 4) determine the core classification tier of the specific diagnostic trouble code:

- `00` → Core / Standardized SAE Code (Universal emissions code uniform across all makes)
- `01` → Manufacturer-Specific Code (Enhanced proprietary code unique to the vehicle brand)
- `10` → Enhanced / Reserved Code Configuration Matrix
- `11` → Circuit / Infrastructure Network Fault Definition

This sub-sector is isolated using a bitwise AND operator with the bitmask `0x3` (`0011`). This logic allows the software engine to resolve the secondary alphanumeric digit (differentiating a standard `P0300` random misfire from a manufacturer-specific `P1300` timing ignition fault), prepping the entity string for exact description mapping. The remaining 12 bits of the structural code block are parsed straight from hexadecimal to standard integers, supplying the trailing three digits of the final trouble string.

---

3. Byte Array Parsing & Multi-Frame Consolidation

3.1. Parsing Padding Tokens and Null-Frame Mitigation

Modern Electronic Control Units transmit information over physical layers in rigid blocks restricted to 8-byte frames. When the number of active vehicle faults does not fill the entire frame capacity, the remaining data positions are filled by the vehicle's firmware using standardized **padding tokens** (typically `0x00`).

The loop inside `DtcDecoder` includes a defensive filter to handle these null frame sectors: upon encountering a sequence of four consecutive zero characters (`"0000"`), the logic triggers an immediate `continue` instruction, ignoring the padding blocks and keeping the app from showing false or non-existent system faults.

3.2. Asynchronous Payload Stream Splitting

Because trouble data arrives from the serial RFCOMM link as an asynchronous byte array, the parsing engine must split the data block into logical elements. Once the primary layers of the `ResponseParser` strip the transmission noise introduced by the ELM327 controller (such as newlines `\r` or prompt marks `>`), the clean hexadecimal bitstream is passed forward.
The decoding engine divides the string using a rigid step interval of **exactly 4 hexadecimal characters (2 bytes)** per processing loop iteration, securing a clean data partition without risking bit-shifting alignment faults during stream ingestion.

3.3. Memory Allocation and Collection Pipelines (List Layouts)

Following bitmask verification and conversion, the finalized alphanumeric strings (e.g., `"P0103"`) are routed into a dynamic allocation container of type `ArrayList<String>`. To block data duplication during multi-frame transmission sequences or overlapping multi-node responses, the collection pipeline runs a structural validation gate via `! detectedCodes.contains(fullyQualifiedDtc)` before inserting the item into the array, preserving structural data accuracy for the UI layers.

---

4. UI Layer Integration & Defensive Error Boundaries

4.1. Lifecycle-Safe LiveData State Distribution Model

Once the active diagnostic trouble code array is assembled by the background thread pool, the array is pushed forward using the **Android Jetpack LiveData framework** via the `activeDtcs.postValue(parsedCodes)` mechanism. This architecture handles the thread-shifting operation, routing data blocks away from background execution pools to the Android Main Thread safely. Presentation layout views subscribe to this stream using lifecycle-safe observers that monitor mutations only when the layout resides in an active runtime status, eliminating memory leaks or unexpected background app crashes.

4.2. Handling Intermittent CAN Bus Congestion and Dropped DTC Packets

High-density data traffic spikes on internal vehicle networks can lead to frame delay anomalies or dropped diagnostic frames during fault memory reads. The software architecture mitigates this volatility at the data boundary link: if a dropped frame results in a corrupted or

incomplete buffer string, `ResponseParser.isErrorFrame()` catches the transmission failure immediately, stalls the parsing chain, and preserves the last verified list structure inside the `LiveData` register instead of letting an empty frame clear the UI screen or trigger an application fault.

4.3. Fallback Layout Execution under "NO DATA" Signal Conditions

If a vehicle contains zero active faults, the powertrain computer returns the standard SAE code string `"NO DATA"`. The client platform intercepts this transmission marker as a successful state of vehicle health rather than an exceptional error execution frame. The parsing engine stops gracefully, returns a clean empty list object, and instructs the presentation layer to dynamically display a safe fallback interface showing the user verification message: `"Active Fault Codes: Clear (No DTCs)"`.

Section 2: Milestone F (Brand-Specific PID Handling and UDS Formatting)

1. Unified Diagnostic Services (UDS) Platform Compliance

The implementation of manufacturer-defined parameter handling shifts the diagnostic core from basic OBD-II queries to the **ISO 14229 (UDS)** standard protocol framework. While standard emissions monitoring is limited to single-byte PIDs, advanced automotive diagnostics require direct, deep access to secondary control nodes such as the Transmission Control Module (TCM), Body Control Module (BCM), and Electronic Braking Units (ABS). By validating communication against the UDS infrastructure baseline, the system can execute advanced telemetry requests utilizing **Service 0x22 (Read Data By Identifier)**. This mode scales beyond standard boundaries by employing a rigid 2-byte Data Identifier (DID) configuration, mapping a massive 16-bit address space (ranging from `0x0000` to `0xFFFF`) to track proprietary hardware metrics.

2. Dynamic Request Formatting & Network Addressing

Unlike legislated OBD-II broadcasts that target the generic functional address `0x7DF`, proprietary manufacturer tracking demands node-specific network addressing. The platform deploys a **Dynamic CAN Header Modification Engine** that handles structural modifications by overwriting the physical layer headers at runtime. By utilizing low-level ELM327 Set Header (`AT SH`) instructions, the execution layer intercepts the diagnostic stream to mask and route frames directly to the target ECU (e.g., `AT SH 7E0` for the Powertrain computer or `AT SH 7E1` for the Transmission controller). For payloads returning data volumes that exceed the standard 8-byte CAN frame boundary, the architecture interfaces directly with the **ISO-TP (ISO 15765-2) Transport Layer** protocol embedded in the serial adapter, controlling multi-frame message block synchronization, flow management, and consecutive frame parsing automatically.

3. Advanced Hexadecimal Response Decoding Matrix

When an enhanced parameter frame propagates back from the automotive bus, the system immediately verifies the transaction through a positive response reflection filter. For UDS Service `0x22` queries, a valid acknowledgment mirrors the service byte by adding a `0x40` hex offset, returning a structural **`0x62` reflection token** followed by the requested 2-byte DID (e.g., `62 1A 04`). Once frame validation completes, the stream passes into the **Bit-Level Decoupling Layer**, which executes pointer-based bitwise masking (`&`, `|`) and logical bit-shifting (`>>`, `<<`) matrices to isolate compressed binary statuses across single-bit positions. The parsed integer bits are then processed through multi-coefficient calibration polynomials:

$$\text{Physical Value}=(G\times \text{Raw})+O$$

where G acts as the fractional scaling gain and O represents the signed calibration offset, translating raw hex data arrays into standardized real-world engineering units like `bar`, `grams`, or `degrees`.

4. Fault Containment & Structural Exception Gates

Enhanced manufacturer communication loops introduce increased volatility due to node sleep states or sub-bus congestion. To isolate these faults, the data link manager incorporates a defensive **Negative Response Code (NRC)** parsing gate. If an ECU rejects an unauthorized or out-of-range UDS transaction, it dispatches an ISO 14229 error sequence starting with the hex token **`7F`** followed by the rejected service id and a specific 1-byte NRC code (e.g., `7F 22 31`). The `ResponseParser` intercepts this token at the boundary interface, cancels execution to prevent data corruption, and returns a safe fallback string (`"0"`). Furthermore, the input stream manager enforces strict **250-millisecond hardware timeout boundaries** on all auxiliary node transactions; if a specialized sub-bus controller drops offline, the system drops the failed frame, isolates the timeout anomaly, and increments the scheduling queue to preserve loop continuity.

Section 3: Milestone E (Standard Telemetry Extension for Speed and Temperatures)

1. Protocol Architecture & Compliance

The diagnostic telemetry expansion within this development phase establishes absolute compliance with the **SAE J1979** and **ISO 15031-5** international standards for passenger car diagnostics. These specifications regulate the digital exchange of environmental and powertrain metrics between the vehicle's Electronic Control Units (ECUs) and secondary client diagnostic scan tools. The platform handles **Service 01 (Show Current Powertrain Diagnostic Data)** requests through an optimized sequential transaction pipeline. Rather than flooding the Controller Area Network (CAN) with concurrent queries, the engine sequentially interleaves the requested Parameter IDs (PIDs) using a deterministic **250ms polling loop frequency**, ensuring steady data ingestion without causing internal automotive bus traffic congestion or buffer overflows.

2. Low-Level Bitstream Decoupling & Parsing Algorithms

The processing layers encapsulate isolated decoding strategies to extract real-world physical values from incoming raw hexadecimal chunks:

- **PID 0D (Vehicle Speed):** The transformation evaluates a single data byte (Byte A) returned by the engine controller. The mathematical decoding algorithm follows the standard SAE definition directly:
  $$\text{Vehicle Speed (km/h)}=A$$
  This provides an operational tracking scale from 0 km/h (`0x00`) to 255 km/h (`0xFF`). Frame validation gates verify that the payload starts with the positive reflection prefix **`410D`** and spans a minimum of 6 characters before passing it to the integer conversion engine.
- **PID 05 (Engine Coolant Temperature):** Thermal data is extracted from an unsigned integer byte using a standardized offset translation:
  $$\text{Coolant Temperature (°C)}=A-40$$
  This converts raw integers (0 to 255) into a signed physical scale from -40°C to +215°C. To block bit-sign inversion errors inside the Java Runtime, data blocks are parsed through an explicit unsigned conversion boundary (`Integer.parseInt(..., 16)`).
- **PID 5C (Engine Oil Temperature):** The extraction engine targets the advanced powertrain calibration register using the identical conversion polynomial (A - 40). Due to the lower operational priority assigned to auxiliary registers inside legacy ECUs, the serial interface driver implements a **10ms to 20ms inter-frame delay** directly following a PID 5C dispatch to prevent physical layer bus timeout drops.

3. Multiplexing & Thread-Safe Telemetry Distribution

Because vehicle communication lines over ELM327 RFCOMM links operate under half-duplex hardware constraints, overlapping serial writes will corrupt the data streams. The platform deploys a `ScheduledExecutorService` running a **Cyclic Sweeping Polling Engine** that forces a strict command-response execution pipeline: a complete data exchange cycle (Write → Propagation → Read → Parse) must terminate before the next queued register query can write to the physical stream. Once values are safely parsed, the system shifts data across execution boundaries to the Android Main Thread via the **Android Jetpack LiveData Framework** using `MutableLiveData.postValue()`. This reactive architecture decouples background communication frequencies from graphic layout pipes, keeping the user interface completely fluid.

4. Defensive Boundary Conditions & Exception Containment

To manage electromagnetic interference and electrical fluctuations typical in moving vehicles, the platform implements strict safety gates at the data link boundary:

- **Signal Deficiencies Guarding:** The `ResponseParser` actively scans inbound buffers for standard SAE trouble sub-strings such as `"NO DATA"`, `"CAN ERROR"`, or `"STOPPED"`. If

intercepted, the engine suppresses fatal runtime exceptions, logs the transient anomaly, and substitutes a safe fallback value (`"0"`) to keep the diagnostic stream loop alive.

- **Bitstream Corruption Protection:** Before any type conversion, strings are checked against alphanumeric boundaries (`[0-9A-Fa-f]`). The conversion logic is completely enclosed inside localized `try-catch` structures targeting `NumberFormatException`. Any frame corruption caused by link noise is instantly discarded at the parsing boundary, safeguarding the active background thread pool from unexpected application crashes.
