Technical Specification: Brand-Specific & UDS Parameter Integration (Milestone F)

1. Unified Diagnostic Services (UDS) Platform Compliance

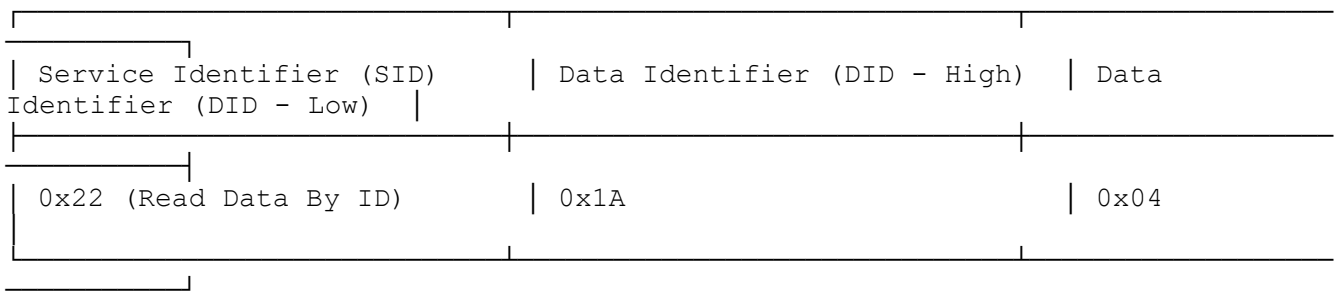
1.1. ISO 14229 Standard Protocol Implementation Baseline

While standard OBD-II (SAE J1979) is restricted to legislated emissions data, the application introduces a dedicated execution base compliant with the **ISO 14229 (UDS)** specification. This international standard governs enhanced manufacturer-specific diagnostics across modern automotive control networks. By establishing a UDS-compliant infrastructure, the application shifts from simple emissions scanning to comprehensive vehicle subsystem interrogation. This allows the core architecture to communicate seamlessly with any electronic control unit (ECU)—such as the Transmission Control Module (TCM), Body Control Module (BCM), or Anti-lock Braking System (ABS)—using unified request-response primitives.

1.2. Mode 22 (Read Data By Identifier) Message Topologies

Under the UDS framework, proprietary real-time sensor parameters are extracted using **Service 0x22 (Read Data By Identifier)**. The structural message topology differs from standard Mode 01 requests by expanding the identifier length. A standard request broadcast drops the traditional single-byte PID layout in favor of a rigid multi-byte payload configuration:

text



Използвайте кода внимателно.

The application's parsing engine dynamically packages this format, ensuring that both the service byte and the full tracking register are properly serialized into the serial transceiver pipeline.

1.3. Dual-Byte Data Identifier (DID) Registry Layering

To support thousands of manufacturer-specific parameters, the data layer implements a **Dual-Byte Data Identifier (DID)** registry system. DIDs are represented as a 16-bit address space (ranging from `0x0000` to `0xFFFF`), which provides the granular routing necessary to isolate proprietary manufacturer routines. The application encapsulates this complexity inside the `BrandSpecificCommand` structure. This module maps each 2-byte DID alongside its unique conversion polynomial, scale factor, and measurement unit. This layered approach allows developers to easily register custom brand configurations (e.g., VAG block measurements,

BMW shadow memories, or Ford extended PIDs) without modifying the underlying network stream buffers.

---

2. Dynamic Request Formatting & Network Addressing

2.1. Dynamic CAN Header Modification Mechanics (SAE J1979 Overrides)

Standard emissions-related OBD-II requests typically utilize a functional broadcast CAN identifier (`0x7DF`), which targets all electronic control units on the vehicle network simultaneously. However, proprietary manufacturer diagnostics require direct, node-specific communication. The application solves this constraint by implementing a **Dynamic CAN Header Modification Engine**. This architecture bypasses standard SAE J1979 broadcast constraints by dynamically switching between 11-bit standard and 29-bit extended CAN identifier layouts at runtime. By updating the network addressing envelope on the fly, the client can shift from multi-node telemetry sweeping to a point-to-point physical connection with a specific sub-bus module.

2.2. Inter-ECU Address Masking and Set Header (AT SH) Instructions

To route enhanced diagnostic frames to specific target computers (e.g., separating Engine management from Transmission or Body controllers), the system uses low-level **ELM327 Set Header (`AT SH`)** instructions. When a brand-specific command is popped from the execution queue, the background worker intercepts the payload and issues a high-priority configuration frame before sending the actual query:

text

```
┌──────────────────────────────┬──────────────────────────────────────────┐
│ ELM327 AT Instruction        │ Network Functional Routing Target        │
├──────────────────────────────┼──────────────────────────────────────────┤
│ AT SH 7E0                    │ Primary Powertrain Control (ECM)         │
│ AT SH 7E1                    │ Transmission Control (TCM)               │
│ AT SH 7E2                    │ Anti-lock Braking System (ABS)           │
└──────────────────────────────┴──────────────────────────────────────────┘
```

Използвайте кода внимателно.

The data layer keeps track of the active network header configuration. It skips dispatching duplicate header commands if consecutive queries target the same node, reducing communication overhead on the serial link.

2.3. Flow Control and Multi-Frame Message Assembly (ISO-TP Transport Layer)

Unlike short standard PIDs, manufacturer-specific UDS parameters often return large data structures that exceed the single 8-byte payload limit of a physical CAN frame. The application relies on the ELM327 processor to manage the **ISO-TP (ISO 15765-2) Transport Layer** mechanics.

When a multi-frame response is triggered, the system handles the data using a strict assembly pipeline:

1. **First Frame (FF) Detection:** The client detects the multi-frame sequence indicator byte returned by the adapter.
2. **Flow Control (FC) Propagation:** The system automatically acknowledges the transmission parameters (Block Size and Consecutive Frame separation timing).
3. **Consecutive Frame (CF) Buffering:** Inbound frames are collected sequentially into an internal software stream buffer until the total expected byte size is reached.

This multi-frame processing architecture allows the app to cleanly ingest complex datasets—such as extended freeze frames, advanced adaptation metrics, and proprietary ECU variant descriptors—without corrupting the telemetry loop.

---

3. Advanced Hexadecimal Response Decoding Matrix

3.1. Positive Response Verification Routing (Service Reflection Code 0x62)

При получаване на входящ пакет от превозното средство през UDS протокола, съответният процесорен слой трябва веднага да потвърди валидността на транзакцията. За разлика от стандартния OBD-II режим (където към сервизния байт `01` се добавя `0x40` и се връща `41`), при UDS **Service 0x22** позитивният отговор от страна на ECU се отразява чрез добавяне на `0x40` към първия байт, което рефлектира в сигнатура **`0x62`**.
Ядрото на `ResponseParser` извършва проверка за наличието на този хедър, последван от оригиналния 2-байтов DID (напр. отговорът за DID `1A04` трябва задължително да съдържа низа `621A04`). Наличието на този валидационен маркер служи като софтуерен портал, който гарантира, че данните не са се застъпили с друг паралелен поток на CAN шината [NGI0 Commons Fund NLnet Foundation, Portable Libre Diagnostic NLnet].

3.2. Bit-Level Bitstream Decoupling and Bitmasking Matrices

Proprietary manufacturer parameters frequently encapsulate multiple distinct vehicle operational statuses within a single byte array to compress network bandwidth. The application introduces a **Bit-Level Decoupling Layer** that goes beyond standard byte boundaries to isolate specific binary statuses:

- **Bitmask Extraction:** Inbound hexadecimal string representations are converted to their raw binary byte components.
- **Logical Masking Operations:** Specialized bitwise operators (`&`, `|`, `^`) isolate predefined bit positions. For instance, evaluating component feedback loops like solenoid statuses or relay engagements utilizes single-bit tracking states (`0` or `1`).
- **Bit-Shifting Alignment:** Isolated bits are shifted logically (`>>` or `<<`) to generate zero-offset binary integer values, making them directly readable by the upper presentation layer.

3.3. Manufacturer-Specific Parameter Scale Conversions and Polynomial Parsing

Once raw data blocks are extracted from the CAN payload buffer, they must be transformed into human-readable engineering units. Unlike standard parameters, brand-specific fields require custom mathematical calibrations defined by **multi-coefficient polynomials**:

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

where G represents the gain factor (fractional multiplier) and O represents the signed calibration offset.

The `BrandSpecificCommand` engine parses the parsed integer bits through a dynamic floating-point matrix. This architectural setup allows the application to cleanly decode highly complex technical parameters—such as target turbocharger boost calculations, diesel particulate filter (DPF) soot concentrations, or steering angle sensor positions—and translate them directly into accurate units like `bar`, `grams`, or `degrees`.

---

4. Fault Containment & Structural Exception Gates

4.1. Intercepting Negative Response Codes (NRC - ISO 14229 Specifications)

When a manufacturer-specific UDS request fails or is rejected by an ECU, the target node does not return a positive reflection frame. Instead, it dispatches a standardized **Negative Response Code (NRC)** frame as mandated by the **ISO 14229** specification. A standard UDS negative response is structurally identified by a 3-byte sequence starting with the hex token **`7F`**, followed by the rejected Service Identifier (e.g., `22`), and terminating with a specific 1-byte NRC fault reason:

text

```
┌──────────────────────────┬──────────────────────────┬──────────────────────────┐
│ Negative Response Prefix │ Rejected Service ID (SID)│ Negative Response Code (NRC) │
├──────────────────────────┼──────────────────────────┼──────────────────────────┤
│ 0x7F                     │ 0x22                     │ 0x31 (Request Out Of Range)  │
│ 0x7F                     │ 0x22                     │ 0x22 (Conditions Not Correct)│
│ 0x7F                     │ 0x22                     │ 0x7F (Service Not Supported) │
└──────────────────────────┴──────────────────────────┴──────────────────────────┘
```

Използвайте кода внимателно.

The application's `ResponseParser` intercepts the `7F` error signature at the data link boundary. Rather than allowing the malformed buffer to reach the decoding math engines, the parsing layer blocks execution, logs the exact NRC hex code for technical audit trails, and returns a safe fallback string (`"0"`). This isolation layer ensures that unauthorized requests or diagnostic access rejections never corrupt the app's internal runtime variables.

4.2. Handling Target Node Unavailability and Sub-Bus Timeout Boundaries

Because brand-specific diagnostics communicate directly with specialized auxiliary ECUs (such as an AWD controller, suspension module, or electronic steering lock), a physical node may completely fail to respond if it enters a sleep state or experiences a localized hardware fault. To prevent these sub-bus communication drops from freezing the diagnostic scheduling loop, the platform deploys strict **timeout boundaries**:

- **Response Guarding:** The `InputStream` polling mechanism caps its blocking execution window at **250 milliseconds**.
- **Packet Dropout Isolation:** If an auxiliary controller drops offline or fails to return a carriage return delimiter (`\r`) within this timeframe, the hardware interface manager triggers an internal timeout intercept.
- **Loop Recovery Execution:** The system discards the dropped transaction frame, marks that specific sensor node as temporarily unavailable, and automatically advances the `ScheduledExecutorService` queue to the next parameter. This design safeguards the application's runtime stability, ensuring that a physical failure in a single sub-bus module cannot cause a deadlock or crash the entire real-time streaming pipeline.
