RE-SOM UDS Client

Software Design Document

(Version 1.0.0)

|  |  |  |  |
| --- | --- | --- | --- |
| S.No | Prepared By | Review By | Version No |
| 1 | Venu | Hareesha/Aparna | 1.0.0 |
| 2 | Jayanth |  | 1.0.0 |
| 3 | Varshitha | Hareesha/Aparna | 1.0.0 |

**Table of Contents**

[1. Introduction 1](#_Toc47390055)

[2. Software Design Document 1](#_Toc47390056)

[3. Scope Of The Document 2](#_Toc47390059)

[4. Boot Loader Services 3](#_Toc47390065)

[5. TriggerBootloaderService 4](#_Toc47390066)

[6. ECU Firmware Update Process 5](#_Toc47390067)

[7. File Structure 6](#_Toc47390067)

[8. File Description 6](#_Toc47390070)-7

[9. Embedded Software Update Process via UDS Protocol 8](#_Toc47390071)

[10. Execution Process 9](#_Toc47390072)-10

# **Introduction**

## **Acronym and definitions**

|  |  |
| --- | --- |
| Acronyms | Description |
| UDS | Unified Diagnostic Services |
| DoCAN | Diagnostic over CAN |
| AL | Application Layer |
| NL | Network Layer |
| TP | Transport Protocol |

# **Software Design Document**

**Vehicle Data Aggregator**

**FOTA Manager**

**E-COS App**

**Vehicle Data Aggregator**

**UDS**

**( 0x10, 0x27,0x3E, 0x31, 0x34,0x36,0x37,0x11)**

**Reception Stack Layer**

**Strategic Interface Layer**

**Data transform Layer**

**Data Preserve Layer**

**CAN TP**

**Session Layer**

**Transport Layer**

**Network Layer**

**Common Data Receiver**

**CAN DRIVER**

**SPI DRIVER**

**UART DRIVER**

**Workflow:**

The diagram illustrates a sophisticated software architecture that facilitates the flashing of Electronic Control Units (ECUs) through Unified Diagnostic Services (UDS) within a Linux kernel environment, with interoperability catered towards an Android application layer. At the application layer, service requests originate from high-level components such as the Firmware Over-The-Air (FOTA) manager or the Embedded-COS (E-COS) application, depending on the nature of the service requested. For bootloader services, the FOTA manager engages, while application services invoke the E-COS app.

Upon receiving a service request, the UDS core service processes two potential types of input arguments: a UDS flow configuration file and a hexadecimal file. The hexadecimal file is necessary only for bootloader services, while application services require solely the UDS flow configuration.

The UDS core service analyzes the UDS flow configuration to determine the sequence of diagnostic services to be requested, such as services 0x10 (Diagnostic Session Control), 0x11 (ECU Reset), 0x27 (Security Access), among others. These service requests are conveyed through the CAN Transport Protocol (CAN TP), which subsequently interfaces with the Common Data Receiver.

The Common Data Receiver serves as the nexus for data transactions, dispatching request data downstream to the UART driver. The UART driver maintains a physical connection with the Body Control Module (BCM) utilizing the UART protocol. Responses from the BCM are then relayed back through the UART driver to the Common Data Receiver.

The Common Data Receiver performs arbitration to determine the appropriate recipient of the incoming data—be it the UDS service or the Vehicle Data Aggregator, which handles data processing through its Reception Stack Layer, Strategic Interface Layer, Data Transform Layer, and Data Preserve Layer. The Vehicle Data Aggregator component is crucial for consolidating and managing vehicle data efficiently.

This intricate exchange ensures that the UDS service and the Vehicle Data Aggregator have the requisite data to perform their designated functions within the vehicular communication ecosystem, thereby enabling effective ECU flashing and other diagnostic services.

# **Scope of the Document**

The document specifies the steps to be followed for the integration of ISO15765 stack along with session layer to any customer specific platform. The document depicts the files and API’s used inside the stack provided.

The ISO 15765 specifies a transport protocol and network layer services tailored to meet the requirements of CAN-based vehicle network systems on controller area networks as specified in ISO 11898. It has been defined in accordance with the diagnostic services established in ISO 14229-1 and ISO 15031-5, but is not limited to use with them and is also compatible with most other communication needs for in-vehicle networks.

The protocol specifies an unconfirmed communication. The diagnostic communication over controller area network (DoCAN) protocol supports the standardized service primitive interface as specified in ISO 14229-2. This part of ISO 15765 provides the transport protocol and network layer services to support different application-layer implementations such as

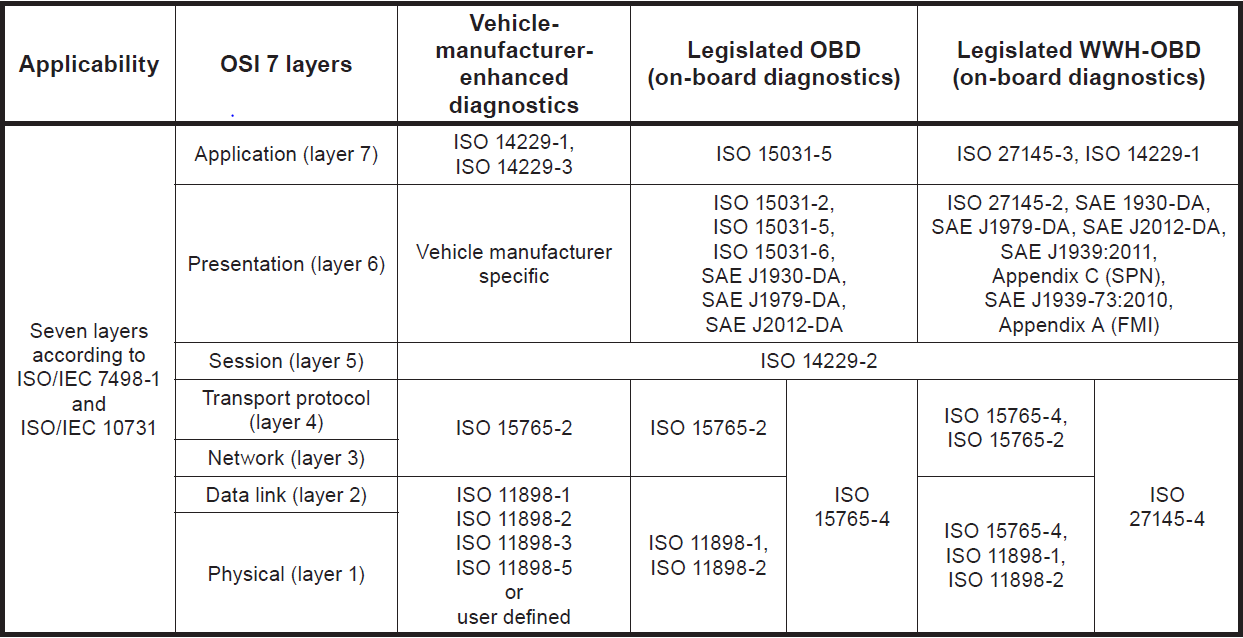
— enhanced vehicle diagnostics (emissions-related system diagnostics beyond legislated functionality, non-emissions related system diagnostics),

— emissions-related on-board diagnostics (OBD) as specified in ISO 15031, and

— world-wide harmonized on-board diagnostics (WWH-OBD) as specified in ISO 27145.

Figure 1 shows Enhanced and legislated on-board diagnostics specifications

applicable to the OSI layers.



# **Boot Loader Services**

**0x10: Diagnostic Session Control**

**0x27: Security Access**

**0x3E: Tester Present**

**0x31: Read Data by Identifier**

**0x34: Request Download**

**0x36: Transfer Data**

**0x37: Request Transfer Exit**

**0x11: ECU Reset**

**Diagnostic Session Control (0x10)**: This service allows the diagnostic tester to establish and control communication sessions with different diagnostic services within the vehicle's electronic control units (ECUs). It enables switching between different diagnostic modes such as default, programming, and extended sessions.

**Security Access (0x27)**: This service provides a means for the diagnostic tester to gain security access to protected functions or sensitive data within the vehicle's ECUs. It typically involves authentication procedures to ensure that only authorized users can access certain diagnostic functions or perform specific operations.

**Tester Present (0x3E)**: The Tester Present service is used by the diagnostic tester to inform the vehicle's ECUs that it is still actively connected and ready to communicate. It helps prevent communication timeouts by regularly notifying the ECUs of the tester's presence during diagnostic sessions.

**Read Data by Identifier (0x31)**: This service enables the diagnostic tester to request specific data values from the vehicle's ECUs using identifiers. It allows retrieving various types of diagnostic information such as sensor data, fault codes, and vehicle parameters stored within the ECUs.

**Request Download (0x34)**: Request Download service is used to initiate the process of transferring data from the diagnostic tester to the vehicle's ECUs for functions such as software updates, parameter settings, or configuration changes. It sets up the parameters for data transfer, including the memory address and length of the data.

**Transfer Data (0x36)**: This service facilitates the actual transfer of data from the diagnostic tester to the vehicle's ECUs, which was initiated using the Request Download service. It manages the transfer process, ensuring data integrity and completeness during transmission.

**Request Transfer Exit (0x37)**: This service helps to manage the orderly conclusion of data exchanges between the diagnostic tool and the electronic control unit (ECU) , ensuring that the ECU recognizes the end of the transfer operation and can resume normal operation.

**ECU Reset (0x11)**: The ECU Reset service allows the diagnostic tester to request a reset or restart of specific ECUs within the vehicle's network. It can be used to clear fault codes, reset adaptations, or initialize ECUs after certain maintenance or diagnostic procedures.

# **TriggerBootloaderService**

Method Name: triggerBootloaderService

Purpose: To initiate the bootloader service for the UDS client application.

Parameters:

* xmlFileUri: URI of the XML configuration file.
* hexFileUri: URI of the hex file for flashing.
* Return Type: Void

**Method Description:**

The triggerBootloaderService method is responsible for initiating the bootloader service, which is essential for updating the firmware of Electronic Control Units (ECUs) using the Unified Diagnostic Services (UDS) protocol.

This method processes the provided XML configuration file and hex file, prepares the necessary data structures, and executes the bootloader service.

**Code Implementation Details:**

Initialization: Initializes the required variables and data structures.

File Handling:

Validates and handles the provided XML configuration file and hex file URIs.

Handles different URI schemes (e.g., content, file).

Template Processing:

Parses and processes the XML configuration file using a template compiler engine.

Hex File Processing:

Converts the hex file to binary format using the HexToBin.convertHexToBinary method.

Service Creation:

Creates instances of UDS services based on the processed template and binary program file.

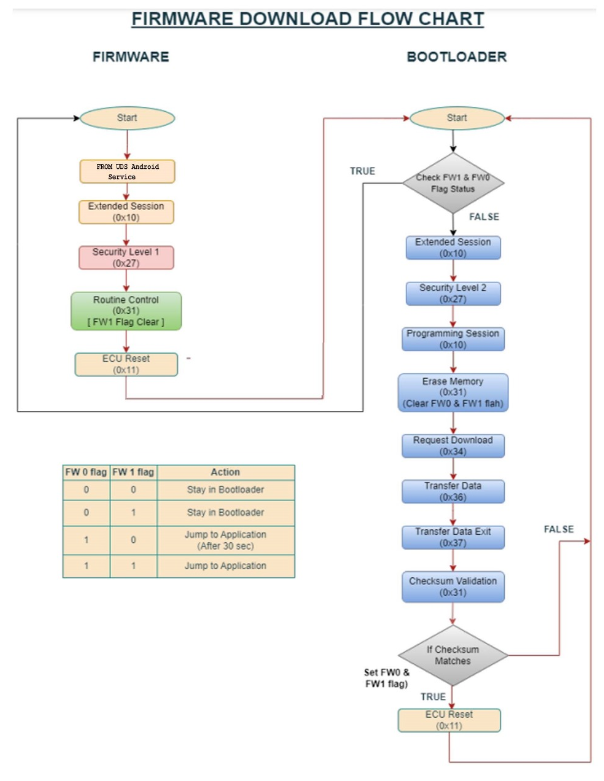
Service Execution:

Submits service execution tasks to an executor service for asynchronous execution.

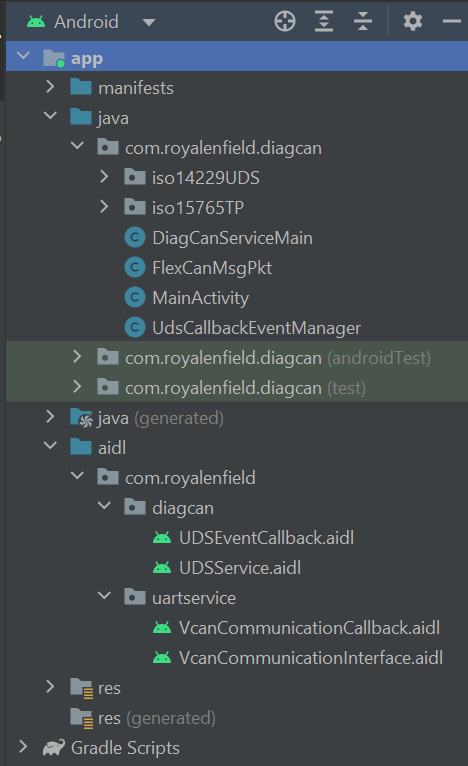
**Thread Execution:**

Submits service execution tasks to an executor service for concurrent execution, ensuring efficient utilization of system resources.

# **ECU Firmware Update Process**



# **File Structure**



# **File Descriptions**

|  |  |
| --- | --- |
| **FILENAME** | **FILE DESCRIPTION** |
| **1. UDSEventCallback(AIDL Interface)** | Defines callback methods associated with Unified Diagnostic Services (UDS) events within an Android application. Specifies an interface for receiving notifications or data related to UDS events. |
| **2.UDSService(AIDL Interface)** | Defines the interface for a service facilitating interactions with Unified Diagnostic Services (UDS) within an Android application. Contains method definitions for clients to interact with the UDS service, such as sending diagnostic requests or configuring settings. |
| **3.VcanCommunicationCallback(AIDL Interface)** | Defines callback methods related to Virtual CAN (VCAN) communication events within an Android application. Specifies an interface for receiving notifications or data concerning VCAN communication events. |
| **4.VcanCommunicationInterface(AIDL Interface)** | Defines the interface for a service responsible for facilitating communication over the Virtual CAN (VCAN) bus within an Android application. Includes method definitions for sending and receiving messages over the VCAN bus and potentially configuring communication parameters. |
| **5.Template Compiler Engine** | It parses XML templates defining diagnostic services, extracting service information and queuing them for further processing |
| **6.Hex File Reader** | Provides functionality to convert HEX files to binary format within an Android application's ISO 14229 Unified Diagnostic Services (UDS) module, handling file parsing and conversion, including addressing information extraction and output stream generation |
| **7.Diag CAN Service Main** | Android service managing UDS-related functionalities, including bootloader and application services, CAN communication via ISO 14229 and 15765 protocols, and file handling for diagnostics. |

# **Embedded Software Update Process via UDS Protocol**

**Start**

**Get Input Files**

**From User**

**False**

**Start Boot Loader Service**

**Stop**

**Execute UDS Services**

**Process template and Hex file and store it to buffers**

**Validate XML & Hex File**

**True**

**XML & Hex file doesn’t exist**

**Submit to UDS Protocol**

# **Execution Process**

The DiagCanServiceMain class serves as the pivotal component within the DiagCAN application, orchestrating critical functionalities for diagnostic and firmware update operations over the Controller Area Network (CAN) bus. Here's an overview of its key aspects:

**Service Integration:**

* Establishes a binding with the CommonDataReceiverIO service, facilitating seamless UART communication and data reception.
* Manages service connection life cycle events and data reception callbacks through a robust implementation.

**ISO 14229 UDS Integration:**

* Seamlessly integrates BootloaderService and ApplicationService modules to administer firmware updates and diagnostic procedures compliant with the ISO 14229 Unified Diagnostic Services (UDS) standard.
* Coordinates UDS service execution based on XML template parsing and HEX file interpretation, ensuring efficient and standardized operations.

**File Processing:**

* Implements robust mechanisms for parsing XML service definition files and HEX firmware files, crucial for service orchestration and firmware manipulation.
* Executes file existence checks and sets default file URIs, enhancing user experience and system reliability.

**Physical Interface:**

* Implements a reliable physical CAN interface, serving as a cornerstone for CAN Transport Protocol (CanTP) communication.
* Manages transmission and reception callbacks, ensuring seamless data exchange and protocol adherence.

**User Interaction:**

* Initiates user interaction by triggering service requests and service data submissions through purposeful intents, promoting user engagement and system usability.
* Provides default file paths for XML and HEX files, streamlining user interactions and enhancing operational efficiency.

**Error Handling:**

* Implements a robust error logging mechanism and comprehensive exception handling strategy, safeguarding against unforeseen errors during file operations, service execution, and communication protocols.

**Service Binding:**

* Exposes a well-defined binder interface, enabling external components to access the rich functionalities encapsulated within DiagCanServiceMain, fostering modularity and extensibility.