Diagnostic Stack

Balija Pavan Kumar

Contents

[**Acronyms and Abbreviations** 2](#_Toc156306769)

[**1. Vehicle Diagnostics Met AUTOSAR Software Architecture** 4](#_Toc156306770)

[**2.** **Overview of Diagnostic Stack in AUTOSAR** 5](#_Toc156306771)

[**2.1 Diagnostics Communication Manager(DCM)** 5](#_Toc156306772)

[**2.1.1 Diagnostic Session Layer (DSL)** 7](#_Toc156306773)

[**2.1.2 Diagnostic Service Dispatcher (DSD)** 8](#_Toc156306774)

[**2.1.3 Diagnostic Service Processing (DSP)** 8](#_Toc156306775)

[**2.1.4 Low-Level Design of DCM** 9](#_Toc156306776)

[**2.2 Diagnostic Event Manager(DEM)** 10](#_Toc156306777)

[**2.3 Function Inhibition Module (FIM)** 13](#_Toc156306778)

[**2.4 Development Error Tracer (DET)** 13](#_Toc156306779)

[**2.5 Non-Volatile Memory (NVM)** 13](#_Toc156306780)

[**2.6 Diagnostic Configuration In AUTOSAR** 14](#_Toc156306781)

[**1.** **Configuration of DCM Module:** 14](#_Toc156306782)

[**2.** **Configuration of DEM Module:** 14](#_Toc156306783)

[**3. UDS Protocol Stack for Vehicle Diagnostics** 16](#_Toc156306784)

[**Introduction to UDS Protocol** 16](#_Toc156306785)

[**What Is UDS Protocol?** 16](#_Toc156306786)

[**Implementation of UDS on CAN in OSI model** 16](#_Toc156306787)

[**ISO-14229 Standards Available** 17](#_Toc156306788)

[**Why Diagnostic need in a vehicle?** 18](#_Toc156306789)

[**UDS Protocol Addressing Methods** 18](#_Toc156306790)

[**Session Layer Timings In The UDS-standard:** 18](#_Toc156306791)

[**UDS Protocol Frame Format** 22](#_Toc156306792)

[**UDS Services** 25](#_Toc156306793)

[**1.** **Diagnostic and communication management** 27](#_Toc156306794)

[**2.** **Data Transmission** 46](#_Toc156306795)

[**3.** **Stored Data Transmission** 60](#_Toc156306796)

[**4. Input/Output Control** 63](#_Toc156306797)

[**5.** **Remote activation of routine** 65](#_Toc156306798)

[**6.** **Upload/Download** 70](#_Toc156306799)

[**Negative Response Codes (NRC): UDS Protocol** 81](#_Toc156306800)

[**4. Does UDS Protocol Work Differently in AUTOSAR Adaptive Platform?** 87](#_Toc156306801)

**Acronyms and Abbreviations**

|  |  |
| --- | --- |
| **Terms:** | **Description:** |
| API | Application Programming Interface |
| CAN | Controller Area Network |
| Dcm | Diagnostic Communication Manager |
| Dem | Diagnostic Event Manager |
| Det | Default Error Tracer |
| DID | Data Identifier |
| DSD | Diagnostic Service Dispatcher (submodule of the Dcm module) |
| DSL | Diagnostic Session Layer (submodule of the Dcm module) |
| DSP | Diagnostic Service Processing (submodule of the Dcm module) |
| DTC | Diagnostic Trouble Codes |
| ID | Identifier |
| LIN | Local Interconnect Network |
| MCU | Micro-Controller Unit |
| NRC | Negative Response Code |
| OBD | On-Board Diagnosis |
| OSI | Open Systems Interconnection |
| PDID | Periodic Data Identifier |
| PDU | Protocol Data Unit |
| PID | Parameter Identifier |
| RID | Routine Identifier |
| ROE | ResponseOnEvent |
| RTE | Runtime Environment |
| SAP | Service Access Point |
| SDU | Service Data Unit |
| SID | Service Identifier |
| SW-C | Software-Component |
| TP | Transport Protocol |
| UDS | Unified Diagnostic Services |
| BSW | Basic Software |
| CDD | Complex Device Driver |
| CRC | Cyclic Redundancy Check |
| DTR | Diagnostic Test Result |
| DYC | Driving Cycle (OBD Term) |
| ECU | Electronic Control Unit |
| EcuM | Electronic Control Unit Manager |
| FDC | Fault Detection Counter |
| Fim | Function Inhibition Manager |
| FMI | Failure Mode Indicator (SAE J1939) |
| FTB | Failure Type Byte |
| HW | Hardware |
| ISO | International Standardization Organization |
| NVRAM | Non volatile RAM |
| OEM | Original Equipment Manufacturer (Automotive Manufacturer) |
| OS | Operating System |
| RAM | Random Access Memory |
| ROM | Read-only Memory |
| SW | Software |
| PCI | Protocol Control Information |
| NVM | Non volatile memory |

# **1. Vehicle Diagnostics Met AUTOSAR Software Architecture**

In automotive, diagnostics is required to be performed on all ECUs to ensure there is no issue with any electronically controlled component of the vehicle. Any issue encountered by the automotive ECU is stored as Diagnostics Trouble Code (DTC) in the Electrically Erasable Programmable Read-Only Memory (EEPROM). These codes can be retrieved later using an automotive diagnostic tool.

Now the vehicle diagnostics system needs to be implemented in the AUTOSAR architecture and this is what this blog aims to explore.

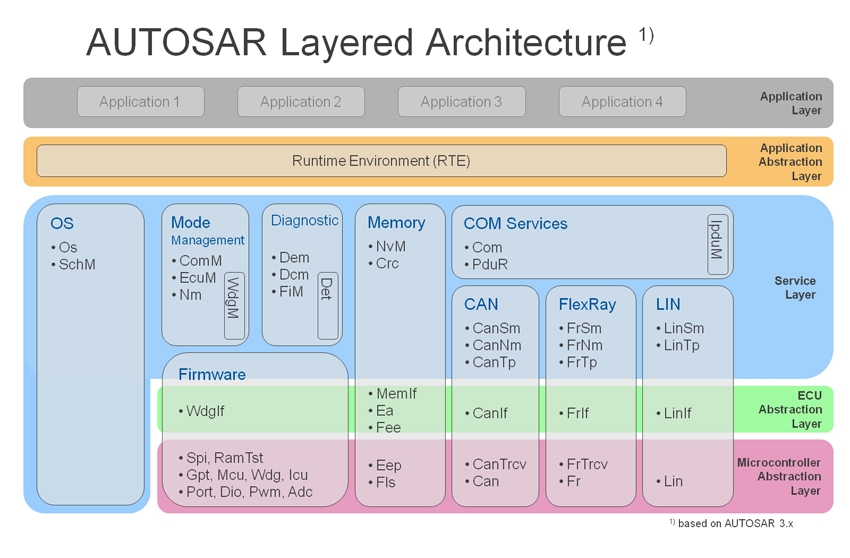
To understand how the diagnostics is implemented in AUTOSAR architecture, let us quickly revisit the architecture.

In the base software layer, there are hundreds of software modules including those categorized under the **Microcontroller Abstraction Layer (MCAL), ECU Abstraction Layers and Service layer**.

The blog will focus on the **service layer** of the AUTOSAR Base Software Module as the vehicle diagnostics services are stored here.

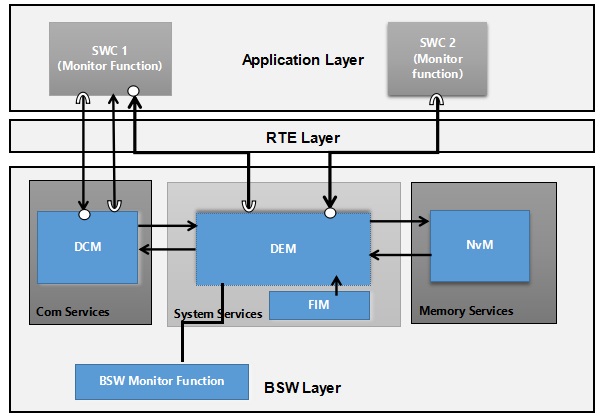
There are essentially three modules inside this layer tasked with different responsibility with respect to vehicle diagnostics. We will discuss them in the subsequent sections.

In the diagram below, the different components of the diagnostic stack**(DCM, DEM, FIM, DET** etc.)



# **2.** **Overview of Diagnostic Stack in AUTOSAR**

Vehicle diagnostics is all about checking the health of your vehicle using some protocols. Protocols are either on board vehicle diagnostics **(OBD)** protocol or off board protocols **(UDS)**. Using these protocols information from different systems or sensors can be read or error reported by ECUs can be read and error related data can be read, and many more things can be done using diagnostics.

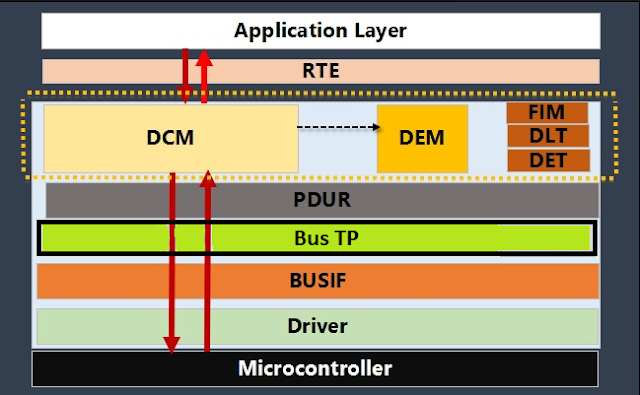


**Fig. 1 AUTOSAR Diagnostic stack**

**Fig. 1** Shows AUTOSAR diagnostics stack overview. Shows different modules of Diagnostics stack and their location in AUTOSAR (Com Service or memory service etc). Diagnostics stack contains DEM (Diagnostic Event Manager), DCM (Diagnostics Communication Manager), FIM (Function Inhibition Manager) and DET (Development Error Tracer).

## **2.1 Diagnostics Communication Manager(DCM)**

The below figure shows how the AUTOSAR DCM module is communicating with other modules in BSW layer.



**Fig. 1 AUTOSAR DCM Block Diagram**

Fig 1 shows shows diagnostic request response flow through AUTOSAR stack. Request from tester tools received by PduR and PduR transmits requested message to DCM module. Then DCM takes action on requested message and communicated with SWC via RTE (if required) or DEM module to read snapshot data in case of $19 or other BSW modules.

**Components Of DCM**

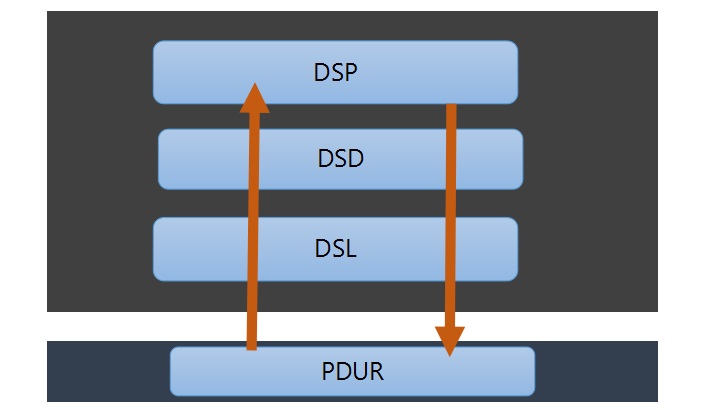
**The diagnostic communication manager (DCM) has the responsibility of reading and writing the fault codes or diagnostic trouble code in the fault memory of the automotive ECU.** It supports the implementation of the diagnostic protocols such as UDS (ISO 14229) and OBD II. When an automotive ECU receives a diagnostics request from the tester tool, the DCM pre-processes it. While it handles majority of the requests, any other request is routed to the functional software. Every new version of DCM has an enhanced functional range which increases its ability to handle diverse types of diagnostic requests

DCM comprises of the service identifiers (SID), data identifiers (DID) sub-functions and routine identifiers (RID) to handle the vehicle diagnostics requests.

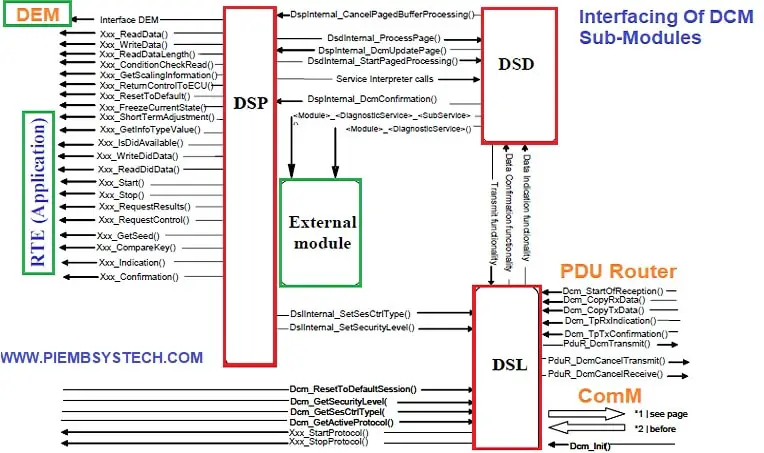
DCM checks security level and session access of requested service. Checks service is supported or not, validates length of requested message. For valid process, invokes certain action based on requested service and provides response to tester tool via PduR.

**DCM module as consisting of the following 3-submodules:**

1. Diagnostic Session Layer (DSL).
2. Diagnostic Service Dispatcher (DSD).
3. Diagnostic Service Processing (DSP).



**Fig.3 DCM divided in sub-modules**



**Fig.4 AUTOSAR DCM Sub-Modules Interface Diagram**

### **2.1.1 Diagnostic Session Layer (DSL)**

This sub-module is tasked with ensuring the flow of data related to the diagnostic requests and the responses**. Diagnostic session and security states are managed by this layer** **along with the supervision of timing parameters of the diagnostic protocol**. DSL interacts with different other modules to achieve its tasks, modules like:

**PduR:** DSL module receives diagnostic requests from PduR and DSL module sends the response for the diagnostic request to PduR

**DSD sub-module:** DSL sub-module informs DSD of incoming diagnostic requests and provides the data. DSL also receives response for diagnostic request from DSD which it forwards to PduR further.

**SWCs or DSP submodule:** DSL module provides access to security and session state.

**ComM module:** DSL sub-module has to take care of accurate communication timings, it achieves it with the help of ComM.

**Let’s study in detail functionalities provided by DSL:**

* Implements concurrent tester present logic
* Reads current security level of ECU. Reads current diagnostic session state of ECU Resets or sets new security level and session state of ECU
* Keeps track of current session, if current session is not default session (i.e. it may be programming or default session) then checks for server timeout and if server timeout occurs resets the session to default session.
* Handling of different diagnostic protocol. i.e. DSL distinguishes between OBD or UDS protocol, so you can configure OBD or UDS protocols, or UDS protocols with different sets of services and and assign them to different diagnostic protocols.
* DSL is responsible for NRC 0x78
* Protocol timings can be modified (P2, P2\*)
* Forward response from DSD sub-module to PduR.

### **2.1.2 Diagnostic Service Dispatcher (DSD)**

The DSD sub-module is responsible for checking the validity of an incoming diagnostic request. Validity here means **verification of Diagnostic Session or Security Access levels or Application permission. This validation helps in processing only valid requests and rejecting invalid requests.** DSD sub-module also keeps track of ongoing diagnostic request processing.

**DSD sub-module interacts with other sub-modules as followed:**

**DSL:** DSL calls the DSD sub-module when received a new diagnostic request message. DSD then forwards this request to DSP and keeps track of ongoing request processing. It also transmits the response of DSP to DSL. DSD sub-module also calls DSL to get latest diagnostic session and security state. DSD module also get the confirmation of transmission response message from DSL.

**DSP:** DSD delegates the received diagnostic request (if valid) and also sends confirmation of transmission of response message to DSP. DSP module sends signal to DSD after processing of diagnostic request is done.

**Let’s understand the functionalities provided by DSD sub-module:**

* Checks Security and Session access of incoming request. i.e. if ECU is in default session and request should be executed in extended session then DSD sub-module sends NRC 0x7F (Service not supported in actvie session)
* Sends positive response or send negative response or suppress response.
* Checks whether service is supported or not. (If service 0x22 is not configured and tester tool sends 0x22 service then DSD sub-module should check service table for supported services and sends NRC 0x11 if service is not supported)
* DSD sub-module collects positive or negative response from DSP sub-module, adds diagnostic service identifier and response data returned by application and forwards diagnostic response message to DSL sub-module

### **2.1.3 Diagnostic Service Processing (DSP)**

The Diagnostic Service Processing sub-module does the actual diagnostic service work task. The Diagnostic Service Processing receives the service requests from the Diagnostic Service Dispatcher sub-module**. Then it performs a check and executes a particular action based on the type of request. It basically checks what is the service identifier and what type of sub-function. So that it can acquire data or execute a required function in Diagnostic Event Management (DEM) or SWCs.** Once it is completed, then it will assemble the response and sends it to the Diagnostic Service Dispatcher sub-module.

**Let’s understand the functionalities provided by DSP sub-module:**

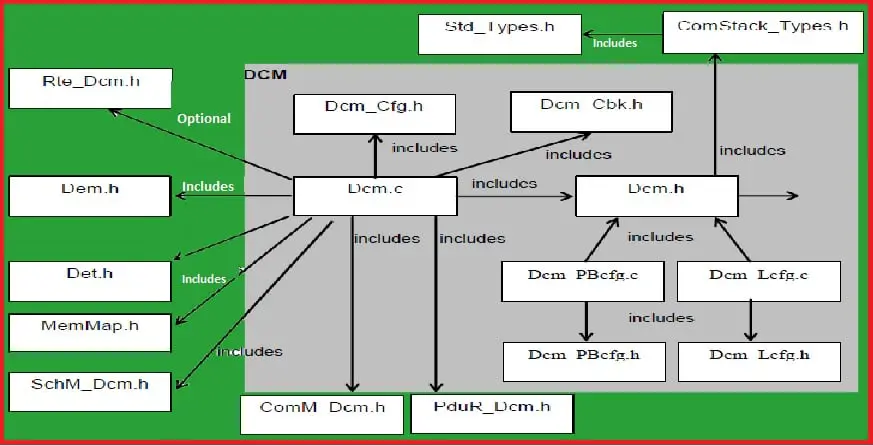
* Once message from DSD is received check for message format (NRC 0x13) check subfunction supported before execution of particular function. (Checks Sub-function is supported or not. Checks against service table and checks sub-function is present in-service table or not. Sends NRC 0x12 sub-function not supported)
* The DSP sub-module shall assemble the response message excluding response service identifier
* Service implementation (i.e. UDS service implementation e.g. implementation service 0x19 or 0x 10 as per ISO14229)

### **2.1.4 Low-Level Design of DCM**

Low-level design is done by drawing the activity diagram or flow chart of APIs realized by the Diagnostic Communication Manager. It defines the internal logic of the APIs. The flow chart for some of the APIs that are realized are given below:

The configuration of the In AUTOSAR DCM module can be done by using different configuration tools like Davinci Configurator. The routing table is configured during the post-build time and the parameters corresponding to minimum routing are configured at link time.

This tool is developed by Vector, in order to build AUTOSAR compliant software for an ECU. The developer has to depend on configuration tools available in the market since the manual configuration is time-consuming. Moreover, each OEM would be having a specific requirement that needs to be achieved. These requirements can be achieved by extending the standard AUTOSAR specification like adding vendor-specific modules, containers, parameters, etc.



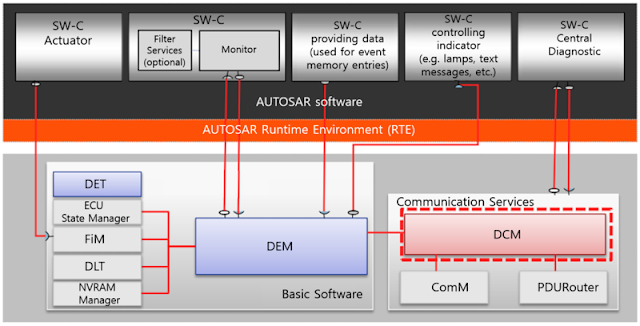
The **Dcm.h** file includes general Diagnostic Communication Manager (DCM) definitions. All the Type definitions of the DCM are defined in **Dcm.h** file. The Pre-compile-time configuration data of the DCM module is defined in **Dcm\_Cfg.h** file. The **Dcm\_.h** file includes the declaration of APIs of DCM used by different modules. The **\_cbk.h** file declares the data types and APIs of other modules used by the DCM module. The **Det.h**, **Dem.h**, and **Dem\_IntErrld.h** files declares the data types and APIs of DET and DEM used by the DCM module.

The **ComStack\_Types.h** contains all types that are used by different modules of the communication stack of the basic software that is platform and compiler independent and **Std\_Types.h** contains all types that are used by other modules of the basic software that are platform and compiler independent. The **Compiler.h** and **Platform\_Types.h** file declares all the types used by different modules that are compiler and platform dependent. **Dcm.c** file defines all the APIs realized by the In AUTOSAR DCM module and its header file structure is given in Figure.

## **2.2 Diagnostic Event Manager(DEM)**

Diagnostic Event Manager (DEM) is part of service layer of AUTOSAR architecture. **DEM module is responsible for diagnostic event processing and storing of event related data to memory and read event related data from memory and provide information to DCM module.** UDS service no. $19 is completely related to DEM module.

Software Components (SWCs) and BSW modules use DEM APIs to report an event status i.e. event is passed or failed. DEM module and interfaced with other SWC and BSW modules. DEM communicate with SWCs, NVM module and DCM module, FIM, DLT, DET.



SWCs communicate with DEM module through RTE, using Client Server port or Sender receiver port. SWC communicates to report status of event to DEM and DEM reads data from SWCs to collect snapshot data. DCM module communicates with DEM and vice versa using AUTOSAR interface to read Diagnostic information (i.e. Service No. 19 Read DTC status or Read Snapshot data). DEM communicates with NVM using AUTOSAR interface. DEM uses NVM to read and write diagnostic information into memory. Other BSW modules communicates to DEM module to report BSW event status whether event is passed or failed.

DCM and DEM come as part of AUTOSAR configuration. AUTOSAR has defined a standardized format for data that can be used to configure DCM and DEM. The format is called diagnostic extract (DEXT). DEXT is part of the ARXML file and contains information to configure DCM and DEM. Tools like CANdelaStudio can be deployed to create and export DEXT file which in turn can be imported to DaVinci configurator tool to configure DCM and DEM.

**Below are some basic points need to understand to configure DEM module.**

**Diagnostic Event and DTC:**

In this section we will see what diagnostic event is and what is diagnostic trouble code. Consider a software component which monitors a health of battery (monitored by monitor function). One event diagnostic event is associated with battery health. Monitor function will report a status of event to DEM. Event id is assigned to an event. This event id is unique per event. SWCs or BSW modules reports event status to DEM using event id. e.g. event id 27 is associated with battery health event. Event ids are assigned by DEM module.

Diagnostic trouble code (DTC) is associated with an event. When event is reported as passed or failed, status of DTC changes. DCM uses DTC number to read information about diagnostic event and DTC (e.g. 19 02 or 19 04 service). e.g. DTC C08088 is assigned to event battery health. As per UDS (ISO 14229) protocol, DTC status is stored in a byte called DTC status byte.

In short, monitor function (a runnable entity in SWC) monitors a event and reports event status to DEM module (event passes/failed) through RTE call. Based on event status DTC status changes.

AS per ISO14229-1 for each diagnostic event, DEM module maintains a UDS DTC status byte information. DTC number is of 4 bytes. There are globally common DTC numbers and supplier specific DTC numbers.

**Event Debouncing:**

**There are three types of event debouncing mechanism provided by DEM module.**

**1. Time Based Debouncing**

**2. Counter Based Debouncing**

**3. Monitor Internal Debouncing**

Time based, and counter based debouncing is implemented by DEM, while monitor internal is implemented by software component. Event is tested and considered as failed if debouncing condition mentioned above (maturation criteria) is satisfied.

**DTC Status byte:**

In ISO14229-1 DTC status byte is defined. Bit 0 to Bit 7 represents DTC status. Test Failed means monitor has evaluated error condition and debouncing is performed (i.e. maturation criteria is met)

Bit 0 : Test Failed

Bit 1 : This Failed This Operation Cycle

Bit 2 : Pending DTC

Bit 3 : Confirmed DTC

Bit 4 : Test Not Completed Since Last Clear

Bit 5 : Test Failed Since Last Clear

Bit 6 : Test Not Completed This Operation Cycle

Bit 7 : Warning Indicator Requested

**Snapshot Data and Extended Data:**

When event is failed then event related data can be stored into NVM. ISO14229-1 defines two kinds of event specific data : 1. Snapshot data 2. Extended data. When battery health monitoring DTC is set, you may want to store battery voltage, battery current, this is what snapshot data.

To store snapshot data and extended data trigger is required. Trigger condition can be when test failed bit is set or confirmed dtc bit is set.

**What is DID?**

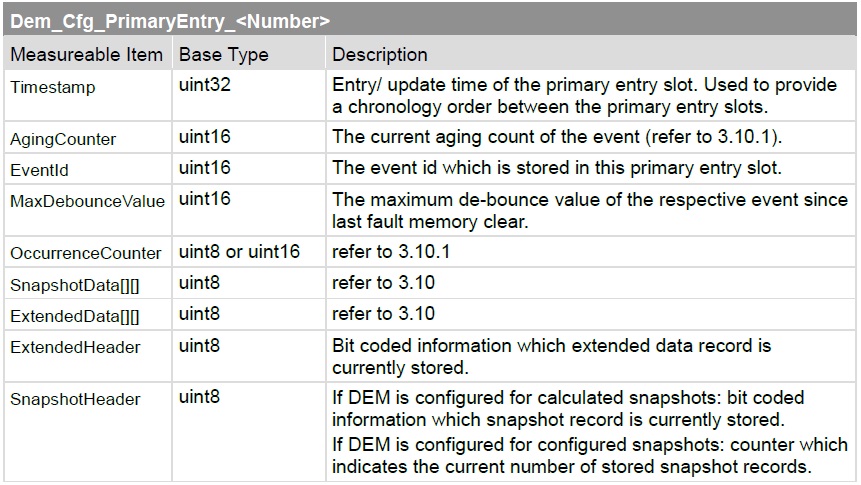
Full form of DID is diagnostic data identifier. Freeze frame/Snapshot data contains a DID or multiple DIDs. DID is a number assigned to identify a data. When DTC is set, and data need to store into NVM. Data is battery voltage and battery current. Then one DID is assigned to battery voltage (e.g. DID FD22) and one DID is assigned to battery current (e.g. DID D025).

Size of DID is user defined. It may be 1 byte or may be 8 bytes. These DIDs are assigned to snapshot data/freeze-frame. Freeze-frame may have a number associated with it (Depends on configuration). DIDs are required for Extended data also.

NVM defined memory blocks for Snapshot data to be stored. RAM blocks for those memory NVM blocks are defined by DEM as Dem\_Cfg\_PrimaryEntry\_0 upto Dem\_Cfg\_PrimaryEntry\_N. N is configurable. Generally configured as 8. While writing to NVM data from RAM block (Dem\_Cfg\_PrimaryEntry\_0) is copied to NV block.

Structure of Snapshot data is shown in below image. This is what gets stored into NVM.

**Note: Below image is taken from Vector Technical Reference manual of DEM. All rights reserved to Vector.**



**Fig. Structure of Snapshot Data**

**Important Point:**

When Snapshot data need to store into memory, from where it is collected? Snapshot data is made-up of DIDs. So, DEM (BSW) will read each DID from application using client server port interface and DEM will act as client and application as server.

To set an event client server port interface is used. DEM acts as server while SWC acts as client. SWC uses DEM API Dem\_SetEventStaus() to report an event status to BSW.

**Event Aging:**

When event is stored into NVM, and event is no longer getting failed, then event starts aging and once event is aged, event is removed from memory. Now how this is done? Consider a event is failed and stored into memory. In an operation cycle event does not failed and event status passed only. Then trigger to aging is started. And aging cycle counter is started and incremented in each operation cycle if event is passed only in operation cycle and does not failed. When aging cycle counter reached threshold value defined by user, event is removed from memory.

## **2.3 Function Inhibition Module (FIM)**

FIM is essentially the control mechanism for AUTOSAR Base Software and software components. The FIM has to control the functionality available to these components depending on their inhibit conditions.An identifier is assigned to the functionalities with an inhibit condition. Only in the scenario of inhibit condition being not true, the functionality is executed.

The role of FIM is to configure and modify the inhibiting conditions of the functionalities. By doing so, a particular functionality can be adapted easily to a new system context.

FIM services are primarily focused on the applications residing in the software components; however, the AUTOSAR Base Software (BSW) can also use the services of FIM when required.

**The FIM has a close connection with DEM as the diagnostic events and their status info are considered as inhibit function by the FIM. When a failure is detected, it is reported to the DEM and it is the job of FIM to stop the particular functionality.**

## **2.4 Development Error Tracer (DET)**

DET is module is used at time of development. While Development DET module is enabled and for final release DET should be disabled. DET module provides APIs to report an error. When DET module is enabled, different checks are added to functions of different BSW modules to capture an error. Error are API function is called with wrong argument, API function is called with NULL Pointer etc. Each error has an error number and each module has a module number.

More information about DET error code for each BSW module is defined in respective SWS document provided by Autosar.

## **2.5 Non-Volatile Memory (NVM)**

NvM is part of memory stack. Here NvM is required to store event related data when event fails. i.e. to store freeze frame data and extended data. Blocks are created in NvM to store diagnostic event related data.

## **2.6 Diagnostic Configuration In AUTOSAR**

### **Configuration of DCM Module:**

**DCM Configuration In AUTOSAR**

The AUTOSAR DCM module is mostly for diagnostic request and response messages. So before you start the configuration in the Basic Software (BSW) module, you should have sound knowledge of UDS or ISO 14229 standard. Next is your system requirements, which means what and all the services, sub-functions, NRC, etc. are supported in your ECU. So that you need to configure it as per this. Basically, you need to configure the DSL, DSD, & DSP.

**Configuration of DSL**

The Diagnostic Session Layer sub-module is responsible for the configuration of the below parameters. It needs to be

* Configuration of Diagnostic buffer for Rx and Tx diagnostic request and response message.
* The Tx and Rx configuration for PduRs.
* Type of protocol and its parameter details configuration.
* All the timing parameter configuration.

**Configuration of DSD**

The Diagnostic Service Dispatcher sub-module is responsible for the configuration of protocol service table. It is having below parameters.

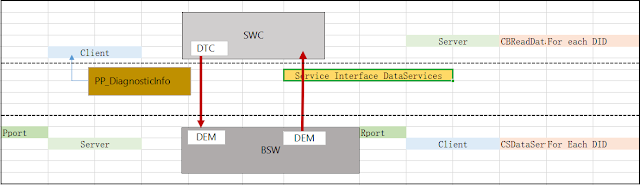
* Configuration of UDS protocol Service Identifiers supported in your ECU.
* Sub-functions of each service supported for this ECU.
* the diagnostic sessions configuration for supporting in your ECU.
* All the security levels and accesses parameter configuration as per your ECU requirement.

**Configuration of DSP**

The Diagnostic Service Processing sub-module is responsible for the configuration of the parameters related to the services. It will configure the configuration related to the service number, sub-function, and data parameters. Suppose for service 0x22, all the supported Data Identifiers (DID) need to configure. You need to configure all the supported DTC for service number 0x19.

### **Configuration of DEM Module:**

Now in this topic we will see if you need to configure DEM module then what things need to configure. Note: Not each and every configuration is mentioned here, but important configuration points considered.



**Fig. DEM Configuration**

As shown in fig.DEM module provides ports called Pport. Using Pport a SWC can set or clear a DTC. For that SWC need to use DiagnosticInfo Port. Where SWC will acts as Client requesting a service to set a DTC and more, while Dem acts as server. To read snapshot data, DEM reads it from SWC, in that case DEM acts as client and SWC acts as server. Here DEM will use Rport Service Interface named DataServices and a callback function (Server) need to implement in SWC, which will prove current value of DID to DEM

**DEM Configuration:**

Remember DEM deal with service 19. i.e DTC, DID, Extended data and Snapshot data, Diagnostic Events. So, while configuration keep in mind you need to configure them. Below are some important points to remember while configuration DEM module:

**1. Configure DTC:**

* Configure DTC number.
* Configure whether DTC is of UDS or OBD type.
* Assign a reference of snapshot data and extended data.

**2. Configure an event:**

* Assign a DTC to an event.
* Enable or disable event aging.
* Aging threshold
* Event debouncing algorithm
* Immediate NVM store option enable or disable.

**3. Configure DIDs:**

* Configure DID number and Name
* Port for SWC to BSW communication: Client Server or Sender Receiver
* DID size, data type

**4. Snapshot Data and Extended Data:**

* Assign DIDs to respective snapshot data and extended data.
* If snapshot is configured to have number, then assign number to snapshot data
* Assign number to extended data

**5. NVM Block:**

NVM blocks for DEM should be configured in NVM and reference need to provide in DEM.

**6. DEM General Settings:**

* To enable aging
* Trigger to store snapshot data
* Event aging type
* Event displacement policy

# **3. UDS Protocol Stack for Vehicle Diagnostics**

## **Introduction to UDS Protocol**

The UDS Protocol is the latest automotive vehicle diagnostic protocol used to diagnose any vehicle all over the world. This protocol is defined in ISO-14229 standard so that every automobile OEM will follow this standard to provide a very common computer system that can be used to diagnose any vehicle. But before that, I want to clear your doubt what is the difference between the communication protocol and diagnostic protocol.

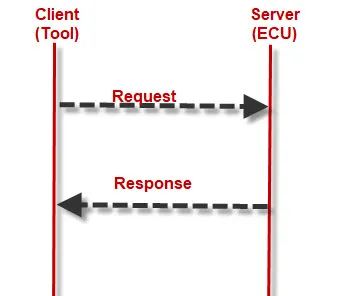
Communication protocol means it is used to communicate between two microcontrollers or to communicate between a controller and a computer to transfer the data. In automotive electronics, we have ECUs (Electronic Control Unit). These diagnostic protocols are used to identify the faults in ECU.

## **What Is UDS Protocol?**

**UDS (Unified Diagnostic Services) protocol is a diagnostic communication protocol used in the automotive industry for diagnosing, testing, and repairing electronic control systems within a vehicle.** It is a standardized protocol that is used to exchange diagnostic and control data between a vehicle’s onboard diagnostic (OBD) system and a diagnostic tool.

UDS is based on the ISO 14229 standard, which defines the services, data types, and communication rules used for diagnostics. The protocol is designed to provide a unified and standardized approach to diagnostics, making it easier for technicians to diagnose and repair different types of vehicles.

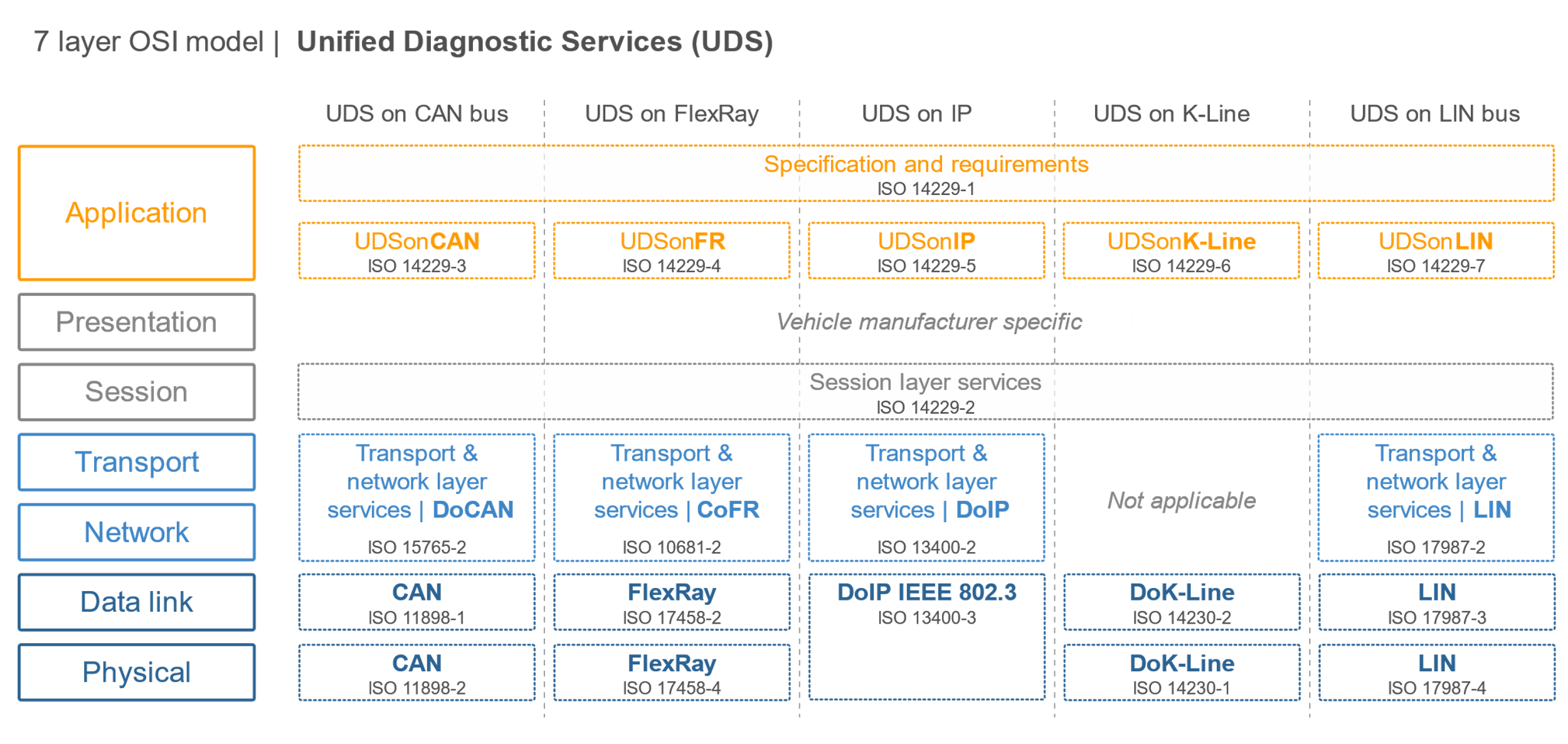
The UDS protocol uses a client-server model, where the diagnostic tool acts as the client and the vehicle’s control module acts as the server. The client sends requests to the server and the server responds with the requested information.



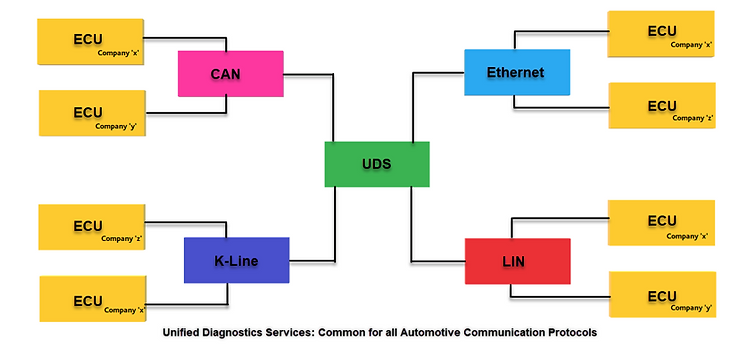
## **Implementation of UDS on CAN in OSI model**

This protocol is defined in ISO-14229-1 standard and it is derived from the ISO 14230-3 (KWP-2000) and ISO 15765-3 (Diagnostic Communication over the CAN (DoCAN).

The UDS (Unified Diagnostic Services) protocol operates at the 5th (Session layer) and 7th (Application layer) of the OSI model. On the other hand, the CAN (Controller Area Network) protocol functions at the 1st (Physical layer) and 2nd (Data Link layer) of the OSI model.



In essence, the UDS protocol builds on top of the CAN protocol to establish communication between a client (requester) and a server (responder) for diagnostic purposes. This mode of communication, where diagnostics are performed on the server upon request, is commonly referred to as Diagnostic over CAN (DoCAN). By utilizing the lower-level functionalities provided by the CAN protocol, the UDS protocol enhances the diagnostic capabilities of the system.



## **ISO-14229 Standards Available**

The UDS protocol specification is defined in different sub-standards of ISO 14229. The ISO-14229Standard UDS Protocol consists of the following parts, under the general title Road vehicles — Unified diagnostic services (UDS):

1. ISO 14229-1: Specification and requirements for UDS Protocol.
2. ISO 14229-2: Session layer services for UDS Protocol.
3. ISO 14229-3: Unified diagnostic services on CAN implementation (UDSonCAN).
4. ISO 14229-4: Unified diagnostic services on FlexRay implementation (UDSonFR).
5. ISO 14229-5: Unified diagnostic services on Internet Protocol implementation (UDSonIP).
6. ISO 14229-6: Unified diagnostic services on K-Line implementation (UDSonK-Line).
7. ISO 14229-7: Unified diagnostic services on Local Interconnect Network implementation (UDSonLIN) (Under ongoing research for implementation).
8. ISO 14229-8: Unified diagnostic services on UDSon… will be prepared gradually and added here

## **Why Diagnostic need in a vehicle?**

When any problem is happening with a human and he is going to the hospital to do the diagnostic and getting a solution, like this for a vehicle also it needs to be diagnostic as to inform the human about his problem such as:

1. We may wish to see data stored within the system – such as Trouble codes – or some form of the identifications.
2. We may wish to see live data – such as the engine or vehicle speed.
3. We may want to transfer a large amount of data – for example re-flashing a module (ECU).
4. We may wish to take direct control of module I/O – for an example disabling individual cylinders to identify the fault.
5. We may wish to run specific routines already in a module – such as some form of self-calibration.
6. We may wish to apply security locks to certain services or to allow the normal function of a system to be disturbed to vary degrees.

## **UDS Protocol Addressing Methods**

To repair, read, write, or flash the new software, the tester needs to connect the testing tool to the ECU. If we want to connect the ECU to the system, we need to assign the address.

There are 2 types of addressing methods.

1. **Physical Addressing**
2. **Functional Addressing**

**Physical Addressing**

In physical addressing mode, If the tester knows which ECU is causing the issue, the tester can connect the testing tool directly to that particular ECU and get the fault code(DTC). To achieve this, each ECU should have its own ECU Identification Number.

So the tester connects with the ECU and sends the request and gets the response from the ECU. This method is called as Physical Addressing method.

**Functional Addressing**

In modern vehicles, a lot of ECUs are available based on the different OEMs. Suppose the tester knows there is a fault in the network(Bus) but he isn’t able to find the exact fault causing ECU in the network. Now, the tester needs to take all the ECUs’ fault codes (DTC) in the network.

In the vehicle, there will be a Global ECU Identifier, which will be implemented in all the CAN receivers. So the ECU can receive the request and give a response to the tester. This method is called as Functional Addressing method.

## **Session Layer Timings In The UDS-standard:**

Contained within the UDS standard there is a standard governing the session layer services in the Open Systems Interconnection (OSI)-model called ISO 14229-2 nothing but UDS Protocol. The most important session layer timings in this project are the P2 and P2 extended timings that specify the maximum time the server (ECU) or client (tester) has to wait or respond to the UDS request.

In the UDS (Unified Diagnostic Services) protocol, the values related to the timing of responses are communicated by the ECU through the UDS Protocol Response to the Diagnostic Session Control service.

For session layer of UDS, there are no special operation beside timing of handling request/response during communication. In general, we have to configured following timer:

**⮚ Tester:**

**P2Client:** Timeout value for the client to wait after the successful transmission of a request message till the start of incoming response message.

**P2Client\_max:** is the maximum value of P2Client

**P2\*Client\_max:** enhanced timeout for the client to wait after the reception of a negative response message, with negative response code 0x78(requestCorrectlyReceived-ResponsePending) for the start of incoming response messages.

**⮚ ECU:**

**P2Server:** performance timer of ECU and is either loaded with P2Server\_max or P2\*Server\_max value.

**P2Server\_max:** server either process the request and send back a response in time or the request processing is still going and the timeout (P2Server\_max value) occur then server send back a negative with NRC=0x78 for “requestCorrectlyReceived-ResponsePending” to notify about pending final response.

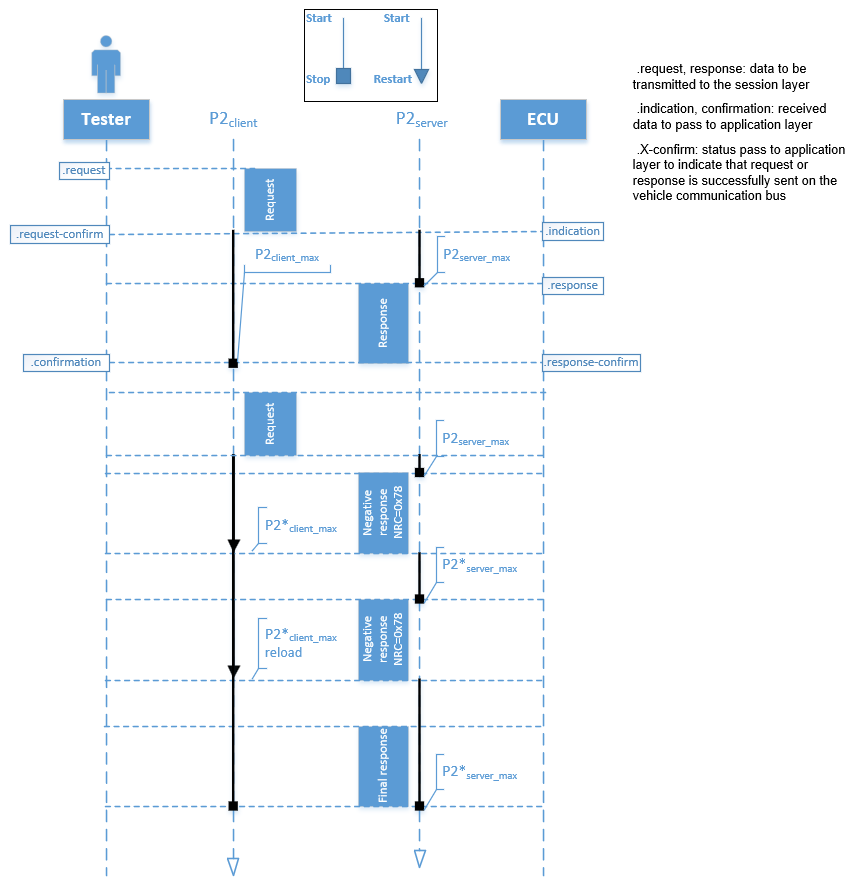
**P2\*Server\_max:** performance requirement for the server to start after the transmission of a negative response message with negative response code 0x78. In case the server can still not provide the requested information within the enhanced P2\*Server\_max, then a further (number of time depend on configuration) negative response message including negative response code 0x78 can be sent by the server.

**P4Server:** is performance requirement time, which is period between the reception of a request and the start of transmission of the final response (which can be either a positive response or negative response with NRC is not 0x78).

In case of a request to schedule periodic responses, the initial Unacknowledged Segmented Data Transfer(USDT) positive or negative response that indicates the acceptance or non-acceptance of the request to schedule periodic responses shall be considered the final response.

If P4Server\_max is the same as P2Server\_max, this means that a negative response with negative response code 0x78 is not allowed for that service or data

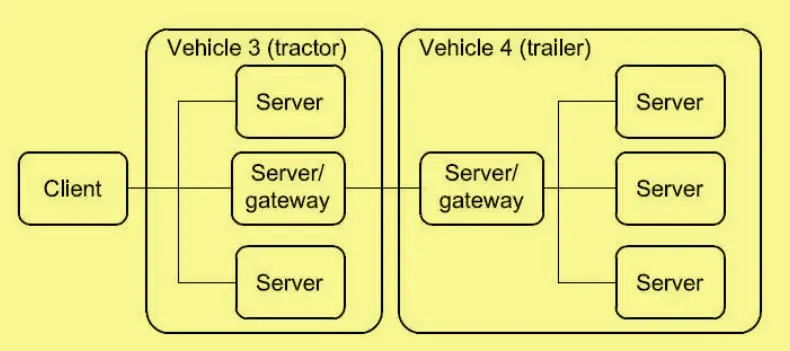
**P4Server\_max:** is the maximum value of P4Server



**UDS Protocol Architecture**

Those services allow a tester (client) to control diagnostic functions in an on-vehicle Electronic Control Unit (server) applied for example on the electronic fuel injection, automatic gearbox, anti-lock braking system, etc., connected on a serial data link embedded in a road vehicle.

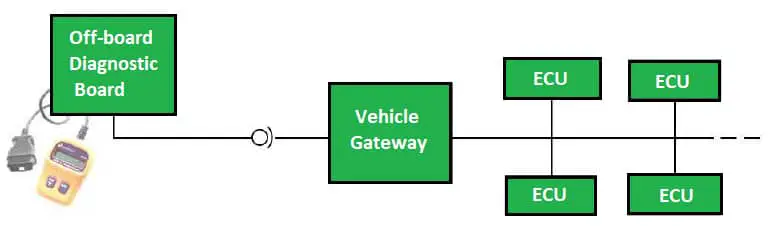
Furthermore, this part of the standard specifies generic services that allow the diagnostic tester (client) to store or to resume non-diagnostic message transmission on the data link. However, part 1 of the standard does not specify any implementation requirements. Figure 7 shows a general configuration of the client-server connection within a vehicle network.



**Client-Server Communication in Vehicle**

For vehicle 3, the servers are directly connected to the diagnostic data link, and vehicle 4 connects its server/gateway directly to the vehicle 3 server/gateway. For vehicle 4, the servers are connected over an internal data link and indirectly connected to the diagnostic data link through the gateways. ISO 14229-1 or UDS Protocol applies to the diagnostic communications over the diagnostic data link; the diagnostic communications over the internal data link may conform to the same or another protocol.

The server, usually a function that is part of the ECU, uses the application layer services to send response data, provided by the requested diagnostic service back to the client. The client is usually referred to as an External Test Equipment when it is off-board but can in some systems, also be an on-board tester. The usage of the application layer services is independent of the client being an off-board or on-board tester. It is a possibility to have more than one client on the same vehicle system.

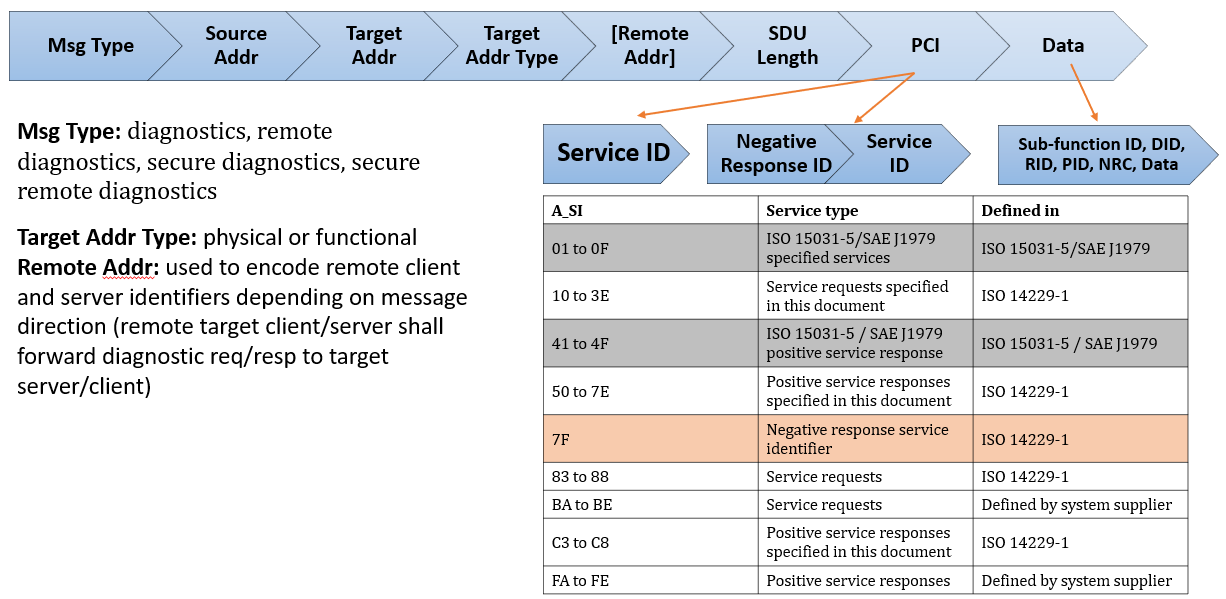


The most typical network configuration of the client-server communication for the vehicle diagnostics: the client as an Off-board tester. Communication is based on a request-response model. In the context of diagnostics, the following concepts are useful for a better understanding of the semantics handled on the UDS standard environment:

1. **Diagnostic Trouble Codes (DTC):** The numerical common identifier fault condition identified by the on-board diagnostic system.
2. **Diagnostic Data:** Data that is located in the memory of an electronic control unit that may be inspected and/or possibly modified by the tester (diagnostic data includes analogue inputs and outputs, digital inputs and outputs, intermediate values and various status information). EXAMPLES: vehicle speed, throttle angle, mirror position, system status, etc.
3. **Diagnostic Session:** The current model of the server, which affects the level of diagnostic functionality.
4. **Diagnostic Routine:** The routine that is embedded in an electronic control unit and that may be started by a server upon a request from the client. NOTE: It could either run instead of the normal operating program or run concurrently to the normal operating program. In the first case, the normal operation of the ECU is not possible. In the second case, multiple diagnostic routines may be enabled that run while all other parts of the electronic control unit are functioning normally.
5. **Tester:** The system that controls functions such as test, inspection, monitoring or diagnosis of an in-vehicle electronic control unit and which may be dedicated to a specific type of operator (e.g. a scan tool dedicated to garage mechanics or a test tool dedicated to the assembly plant agents).

## **UDS Protocol Frame Format**

**Application Protocol Data Unit:**



Since the UDS protocol is working on the CAN protocol so that the maximum 8-bytes of the data can be requested and get to the response in a message. Like CAN protocol, in UDS protocol, there are 2-types of frames are available. The UDS protocol frame format is defined below.

1. **Diagnostic request Frame (With/without Sub-function-ID).**
2. **Diagnostic Response Frame.**

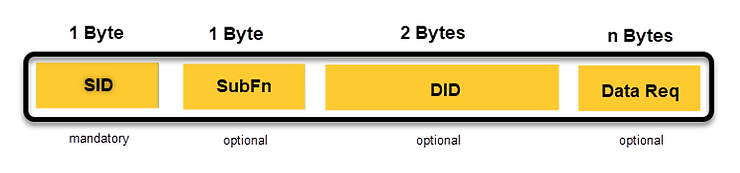
Again, the Response frame is divided into two types:

1. **Positive Response.**
2. **Negative Response.**

**UDS Protocol Diagnostic Request Frame Format:**

Whenever the client wants to request anything from the data then the tester will send this request the frame to get the response from the server on the CAN data field. This frame had consisted of 4 fields:

1. **Service ID.**
2. **Sub-Function ID (optional: not exist for some diag. services).**
3. **DID**
4. **Data bytes.**



**SID or Service identifier:**

This is a 1-Byte hexadecimal code that values from 0x00 to 0x3E. This identifier allows the server (ECU) to understand which service is requested by the client/tester. This is a mandatory field as, without the service identification number, the service request will not be understood by the ECU.

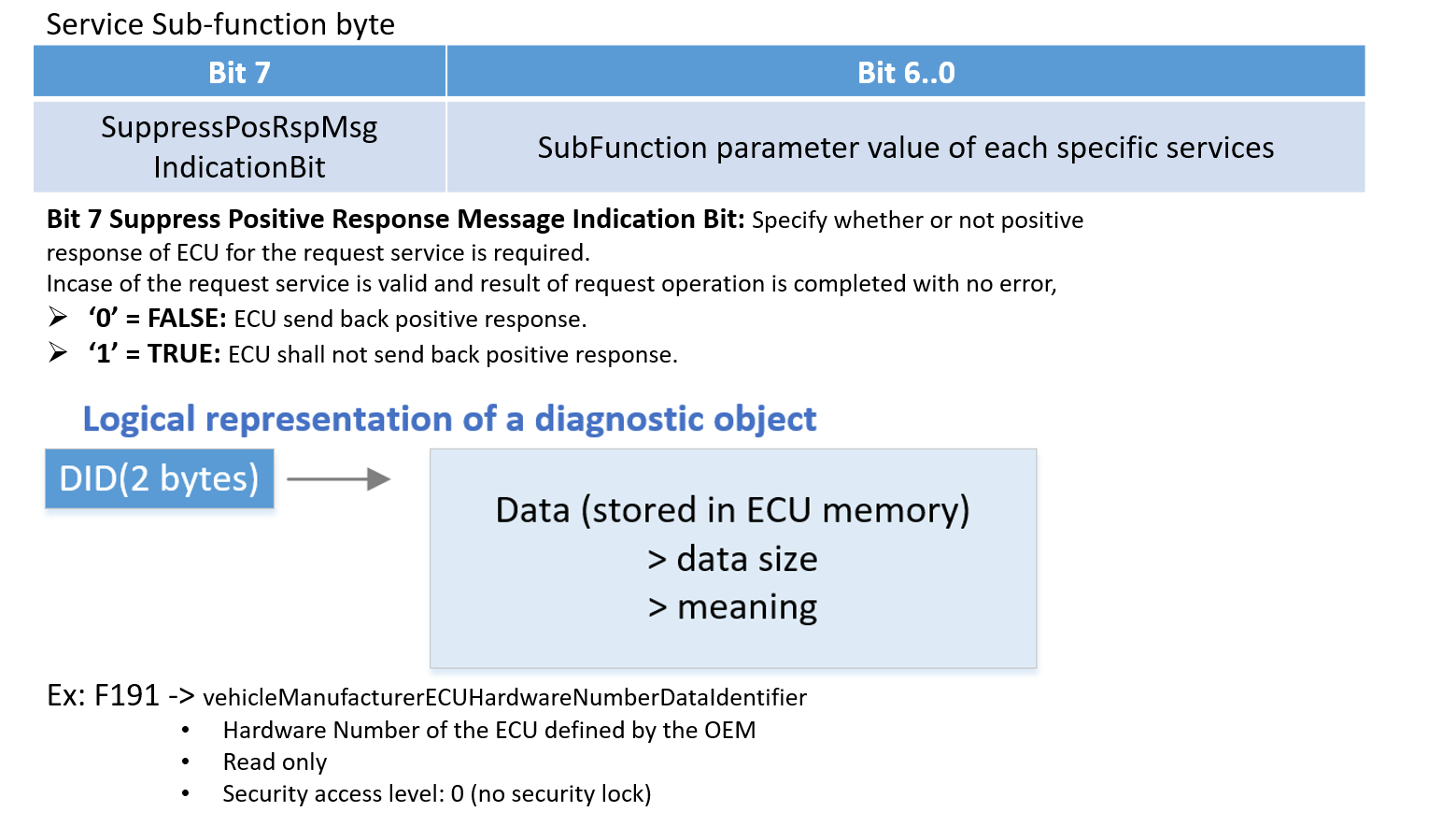
**SubFn or Sub Function:**

tells the server which sub-functionality is requested for a given service. This is also 1-byte long and belongs to a single server. For example,

* 01 - Request service
* 02 - Start service.
* 03 - Stop the service.
* 04 – Request results. Etc.

Sub-functions filed is an optional field as these are needed for some selected services only. Some services that need reading or writing data by the identifier require no sub-functions.

DIDs, PIDs, RIDs are similar since it is a logical presentation of non-volatile data(DID/PID) or function (executable operation RID).



**DID or Data Identifier:** The client and server communicate using only numbers in a basic format. A number must be assigned for all the data elements that the tester reads, known as the data identifier for that particular data element. Did is 2-byte long in UDS, and different elements are identified using different data identifiers by both the server and the client.

* OBD - in Onboard diagnostics, DIDs are standardised globally.
* UDS – Car manufacturers define their own DIDs; only the tester tools from the OEM can read these DIDs

The data identifiers field is completely optional, as there are services that do not need DIDs at all. Or, a single or multiple data identifier can also be present in a single message. These can be Data-oriented (DID), routine Oriented/controlled (RID), MID, or UID.

**Data Req or Data Request Filed:** This field consists of the mete-data related to the data identifier of a specific message. This is n-bytes long and can be optional or mandatory as per the service request.

**UDS Protocol Diagnostic Response Frame Format:**

Whenever a diagnostic engineer or tester will request any service to a vehicle, there is a possibility of two types of response from the vehicle or a particular ECU as per physical or functional request type.

**Positive Response Frame Format:**

Whenever the tester will request to the server if it is correct and the server has been executed the request successfully, then it will send the response message concerning this request by adding 0x40 to the respective service ID for reference. Positive response 1st byte should be Request Service ID + 0x40.



**Negative Response Frame Format:**

If the client did not request in a proper frame format or the server is not able to execute the request due to the internal problem, then it will send a negative response to the client.

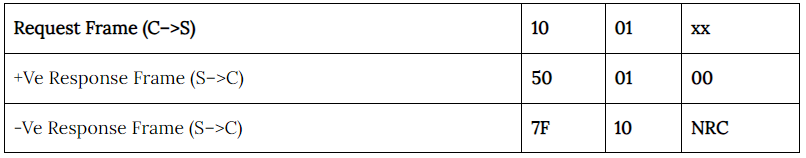
Negative response 1st byte should be 0x7F.

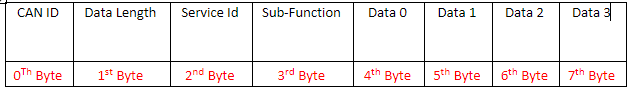
Negative response 2nd byte should be Service ID.

Negative response 3rd byte should be Response Code.



**Example of both +ve and -ve Response:**



**Structure of UDS Frame over CAN:** ****

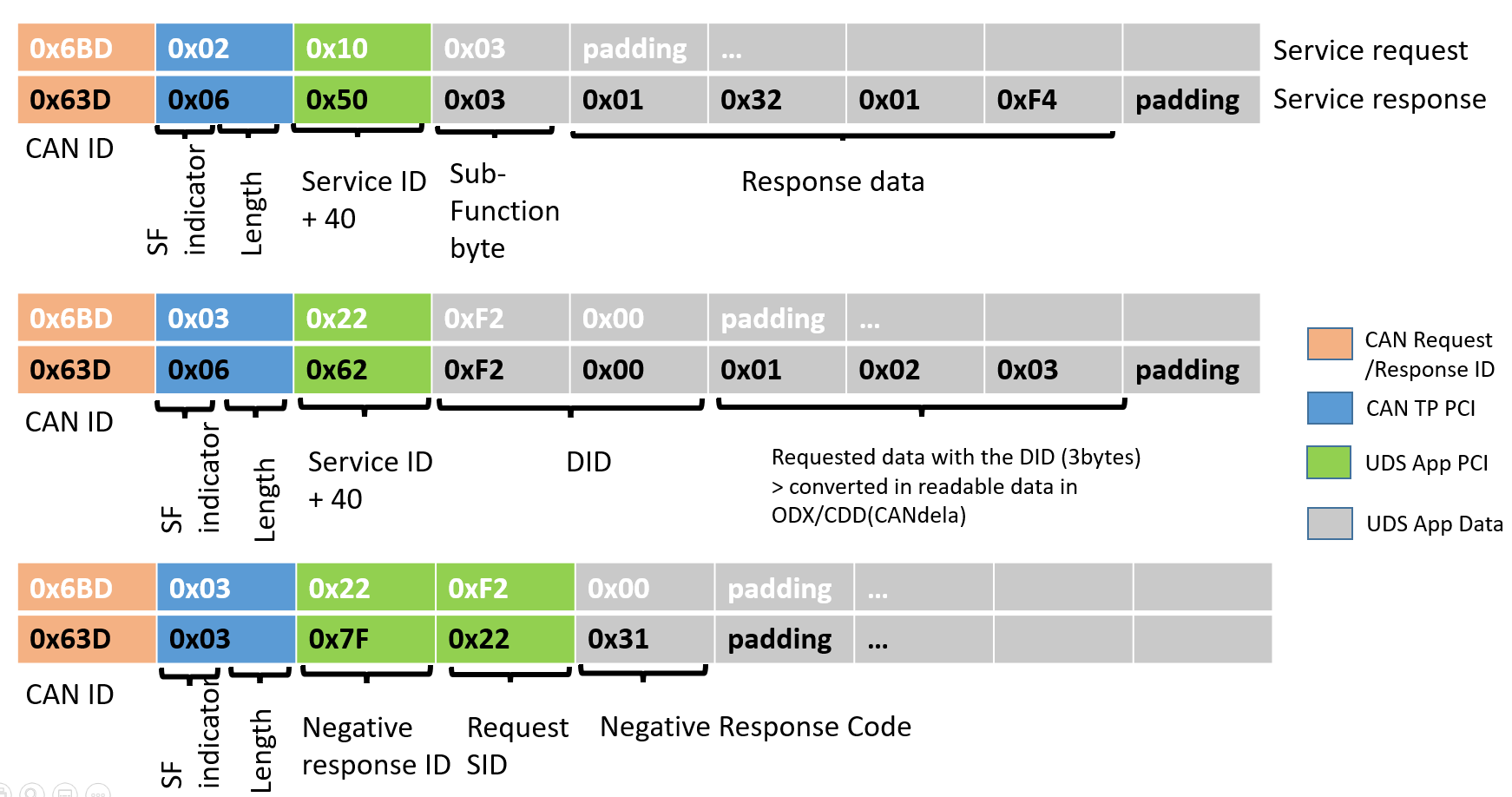
Several UDS services support sub-functions(SF) to further define the request functionality of the SID

➢ e.g. ECUReset, Session Change

Several UDS services support a Data Identifier(DID) to get access to data via a logical number(DID) which is used for the UDS communication

➢ e.g. Read & Write Data

CAN TP PCI (protocol control information) is CAN transport layer PDU header information, specified by ISO 16572-2



## **UDS Services**

Besides specifying services’ primitives and protocols that describe the client-server interaction, UDS also defines within its framework several functional units that comprise several services each, identified with a hexadecimal code. These units are intended for the different individual purposes that support the overall diagnostic function/task. The UDS protocol having different services for the different types of work tasks to do on the server.

**These are having 6- types as:**

1. **Diagnostic and communication management**
2. **Data Transmission**
3. **Stored Data Transmission**
4. **Input/Output Control**
5. **Remote activation of routine**
6. **Upload/Download**

**Brief diagram of all 6 services**

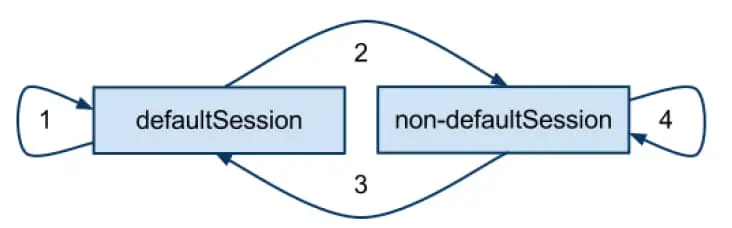


### **Diagnostic and communication management**

There are 10 services are available in this module to control the diagnostic and the communication-related in the ECU.

#### **Diagnostic Session Control (0x10)**

* The diagnostic session’s service ID is 0x10 and the Response SID is 0x50.
* Diagnostic Session Control is one of the most important services used by UDS. Diagnostic Session control is used for controlling the diagnostic session of the ECU. This service is used to change the current diagnostic session to a different session.
* These sessions are used to enable or disable a particular set of diagnostic functions and features in an ECU.
* There are many services available in UDS but all the services cannot access the ECU. Some services can access only if the ECU is in the Default session and some of the other services can access the ECU in other sessions.
* We can say this diagnostics session control is the gateway to access the ECU. Before doing any diagnostics operations this diagnostic service should give access to run the corresponding services. It is decided by vehicle manufacturers.
* Let’s say for example In the default session we can not upgrade the software for the ECU because the Default session doesn’t have access to re-program the ECU. Only in the programming session, we have access to re-program the ECU by the authorized line engineers.
* The main purpose of this diagnostic session is to give security to the ECU. It will prevent the ECU from unwanted access. And only an authorized person can access it. If every diagnostics service can access the ECU, it may get damaged by the wrong flashing of software.
* So before making any diagnostic request, the Client must make sure that this service is accessible or not in the ECU current session. if not, then first send a request for session change after that desirable request shall be performed.
* If the Server(ECU) is in a non-default session or if there is diagnostic inactivity for 5 seconds, the ECU goes back to the default session.
* Basically, this service is used to enable the different diagnostic sessions in the server to work on it. In every session, they have defined some diagnostic services which only enable these sessions so that they will work perfectly without any negative impact on the server.



* If we want to switch from one non-default session to another non-default session, first we need to switch to the default session then we can able to switch to another non-default session.

**This has many sub-functions that have been explained in the below table.**

|  |  |
| --- | --- |
| **Sub-function** | **Description** |
| Default Session(0x01) | * This session is in an idle state. Whenever the ECU is powered ON, it will be in this default session only. ECU remains in default session until another diagnostic session is requested from the client. * Services like Write data byte identifier (0x2E), Read data byte identifier (0x22), ECU Reset (0x11), Tester Present (0x3E), and Reading DTC (0x19) are available in this session. * Security access (0x27) is not available. This service has low security compared to all other diagnostics session control sub-functions.   Example: In a garage, a person is trying to read the Diagnostic Trouble Code(DTC). |
| Programming Session(0x02) | * ECU boot mode for new software flashing. * This diagnosticSession enables all diagnostic services required to support the memory programming of a server. * In case the server runs the programmingSession in the boot software, the programmingSession shall only be left via an ECUReset (0x11) service initiated by the client, a DiagnosticSessionControl (0x10) service with sessionType equal to defaultSession, or a session layer timeout in the server. |
| Extended Session(0x03) | * All services allowed in the Default session are allowed in this session * This session is used to unlock the additional diagnostic functions also * Security access(0x27) is allowed in this session, meaning security levels are unlocked in this session * ECUs entered into default session if this session is ended or expired   Example: This session is End of line engineers doing a dynamic vehicle test to check the security level |
| Safety system Diagnostic system(0x04) | * Used to test all safety-critical diagnostic functions. * This diagnostics session control sub-function has a high level of security.   Example: This session is checking the safety of Airbags, and tire pressure monitors. |
| 0x05 to 0x3F | Reserved |
| Vehicle Manufacture Specific (0x40 to 0x5F) | This session depends on Each OEM, if they want to implement any session based on their requirement they can use it.  Ex: Volvo, Audi, etc. |
| System supplier Specific (0x60 to 0X7E) | This session is also like Vehicle Manufacture Specific only but instead of the vehicle manufacturer, if the system suppliers want to implement any session based on their specific requirement they can use it.  Ex: Robert BOSCH |
| 0x7F | It is still reserved for the future, still, it is not used for any feature |

**Request Frame Format:**

|  |  |  |
| --- | --- | --- |
| **Data byte** | **Parameter Name** | **Byte Value** |
| #1 | DiagnosticSessionControl Request SID | 0x10 |
| #2 | sub-function = [ diagnosticSessionType ] | 0x00 – 0xFF |

**Response Frame Format:**

There are two types of response frames. That is a **Positive response** and a **Negative response**.

* **Positive response**

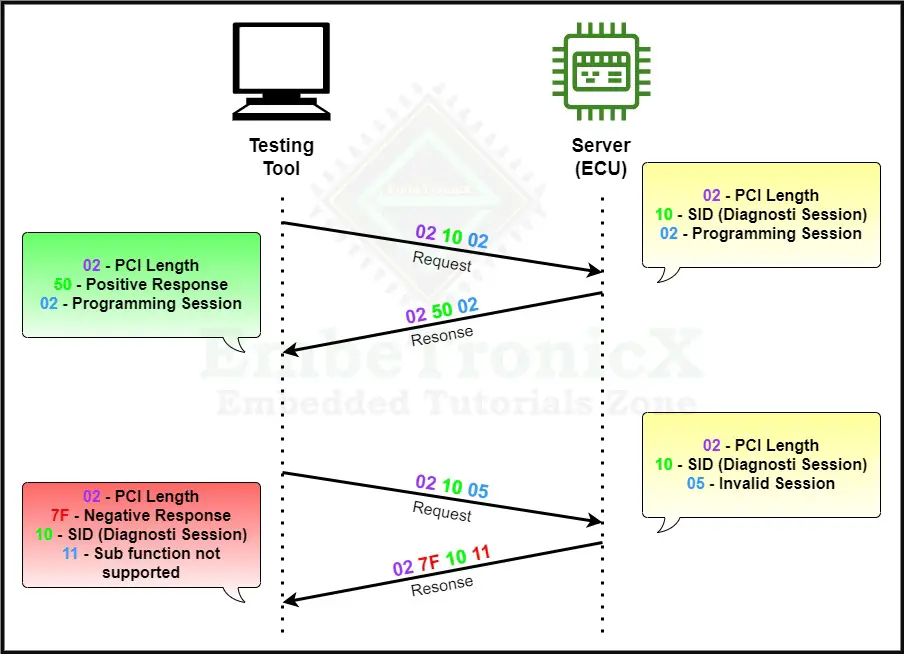
|  |  |  |
| --- | --- | --- |
| **Data byte** | **Parameter Name** | **Byte Value** |
| #1 | DiagnosticSessionControl Response SID | 0x10 + 40 = 0x50 |
| #2 | sub-function = [ diagnosticSessionType ] | 0x00 – 0xFF |
| #3  :  #6 | sessionParameterRecord[]#1 = [  data#1  :  data#4 ] | 0x00 – 0xFF  :  0x00 – 0xFF |

* **Negative response**

The following negative response codes shall be implemented for this service

|  |  |  |  |
| --- | --- | --- | --- |
| **Response ID** | **SID** | **NRC** | **Description** |
| 7F | 10 | 0x12 | **sub-functionNotSupported**  This NRC shall be sent if the sub-function parameter is not supported. |
| 7F | 10 | 0x13 | **incorrectMessageLengthOrInvalidFormat**  This NRC shall be sent if the length of the message is wrong. |
| 7F | 10 | 0x22 | **conditionsNotCorrect**  This NRC shall be returned if the criteria for the request DiagnosticSessionControl are not met. |

**Example:**



#### **ECU Reset (0x11)**

* The ECU reset’s service ID is 0x11 and the Response SID is 0x51.
* ECU Reset service is used to restart the particular ECU or all the ECUs in the Vehicle. The motive of this service is to recover the ECU from malfunction or hanged state or non-working condition.
* During the reset of the ECU, it will not accept any request from the client and will not send any responses to the client. This service is supported in an Extended session.
* After a successful reset, the ECU shall return to the default session.

**This has many sub-functions that have been explained in the below table.**

|  |  |
| --- | --- |
| **Sub-function** | **Description** |
| Hard Reset (0x01) | * Hard reset means removing the battery (power supply) from the ECU and connecting the ECU again with the battery. * In this type of reset, ECU re-initializes the core hardware components of the system and also It will re-initializes the Non-volatile and volatile memory. * This reset may lose some data because the battery is removed suddenly during the running time of the ECU. |
| Key off On Reset (0x02) | * The Key Off-On Reset is simply the ignition Off-On process of a vehicle. It is the normal sleep-wake-up mode of a Microcontroller. * When we are doing Key Off On reset ECUs will not get the power down immediately. It will go to the boot mode then it will store all the data in Non-Volatile memory and de-initialize the hardware variables without losing any data. * In this method of resetting, there is no chance of losing data. This is the proper way to reset, and most of the OEMs are using this type of ECU Reset. |
| Soft Reset (0x03) | * A soft reset is nothing but restarting the application’s main software. When we do this type of reset, the stack pointer of the microcontroller points to the main() function’s address. Then it will start to execute from first.   For Example, will take a watchdog reset. Whenever hanging or any malfunctions are happening, this watchdog timer will reset the microcontroller and it will start from the main() function. |
| Enable Rapid Power shutdown (0x04) | * In this type of reset, ignition off will not occur. ECU will go to sleep mode and ECU is ready to wake up at any time. |
| Disable Rapid Power shutdown (0x05) | * This service is used to disable the previously enabled rapid power shutdown. |

**Request Frame Format:**

|  |  |  |
| --- | --- | --- |
| **Data byte** | **Parameter Name** | **Byte Value** |
| #1 | ECUReset Request SID | 0x11 |
| #2 | ResetType | 0x01 – 0x05 |

**Response Frame Format:**

There are two types of response frames. That is a **Positive response** and a **Negative response**.

* **Positive response**

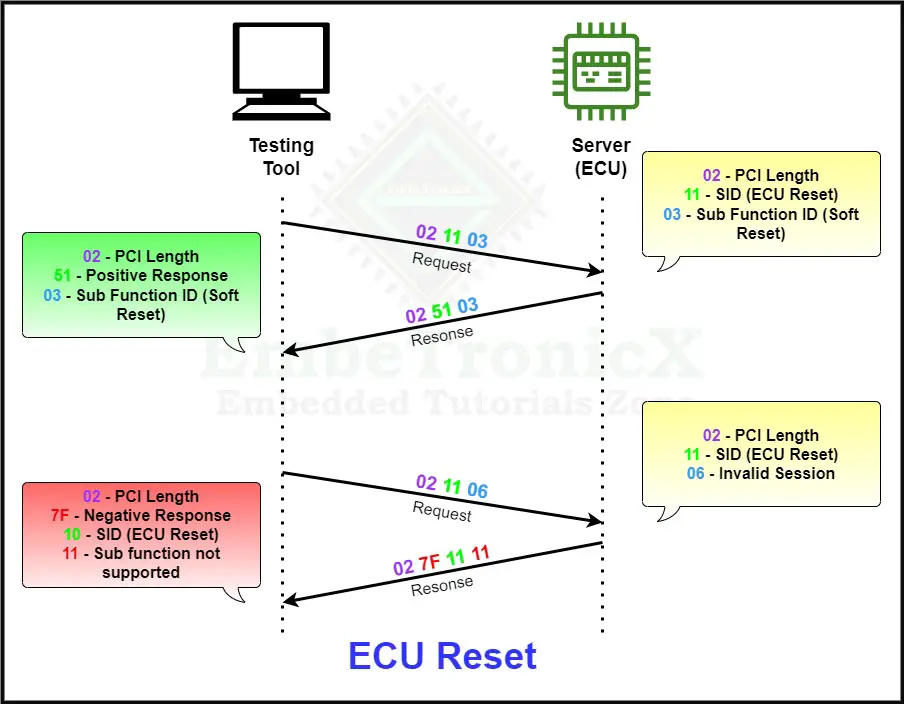
|  |  |  |
| --- | --- | --- |
| **Data byte** | **Parameter Name** | **Byte Value** |
| #1 | ECUReset Response SID | 0x11 + 40 = 0x51 |
| #2 | ResetType | 0x01 – 0x05 |

* **Negative response**

The following negative response codes shall be implemented for this service

|  |  |  |  |
| --- | --- | --- | --- |
| **Response ID** | **SID** | **NRC** | **Description** |
| 7F | 11 | 0x12 | **sub-functionNotSupported**  This NRC shall be sent if the sub-function parameter is not supported. |
| 7F | 11 | 0x13 | **incorrectMessageLengthOrInvalidFormat**  This NRC shall be sent if the length of the message is wrong. |
| 7F | 11 | 0x22 | **conditionsNotCorrect**  This NRC shall be returned if the criteria for the request DiagnosticSessionControl are not met. |
| 7F | 11 | 0x33 | **securityAccessDenied**  This NRC shall be sent if the requested reset is secured and the server is not in an unlocked  state. |

**Example:**

****

#### **Security Access (0x27)**

* The Security Access’s service ID is 0x27 and the Response SID is 0x67.
* This security access diagnostic service is used to give security access to the UDS protocol services to avoid security breaches. Some diagnostics data and services can be restricted for safety purposes.
* This service will grant Read and Write access to the particular service. Before processing any of the Service IDs(SID) operation, it’s mandatory to check the security access, to know whether, for this SID, the client has access to read or write.
* For example, to upload and download the software, routine control and VIN (Vehicle Identification Number) DID cannot be accessed by everyone some authorized Testers can only access these SIDs.

**Security access is granted based on these two,**

1. **Seed (0x01)**
2. **Key (0x02)**

* The tester sends the request to unlock the ECU using SID 0x27 and sub-function ID 0x01. 0x01 means requesting for seed.
* The UDS server receives the request assumes conditions are correct and generates the random seed and key based on a cryptographic algorithm and the Server sends the Seed to the client with a positive response.
* With the received seed, the tester tool generates the key and sends this key to the server to unlock the ECU using SID 0x27 and sub-function ID 0x02. 0x02 means key.
* If the unlock key sent by the tester tool(client) matches with the server expecting key it will send the positive response and Unlock the ECU otherwise it will send a negative response with the specific negative response code.

**This has many sub-functions that have been explained in the below table.**

|  |  |
| --- | --- |
| **Sub-function** | **Description** |
| requestSeed (0x01) | RequestSeed with the level of security defined by the vehicle manufacturer. |
| sendKey (0x02) | SendKey with the level of security defined by the vehicle manufacturer. |
| requestSeed (0x03, 0x05,  0x07 – 0x41) | RequestSeed with different levels of security defined by the vehicle  manufacturer. |
| sendKey (0x04, 0x06,  0x08 – 0x42) | SendKey with different levels of security defined by the vehicle  manufacturer. |

**Request Frame Format:**

* **sub-function = requestSeed**

|  |  |  |
| --- | --- | --- |
| **Data byte** | **Parameter Name** | **Byte Value** |
| #1 | SecurityAcces Request SID | 0x27 |
| #2 | sub-function = [ securityAccessType = sendKey ] | 0x01, 0x03,  0x05,  0x07 – 0x7D |
| #3  :  #n | securityAccessDataRecord[] = [  parameter#1  :  parameter#m ] | 0x00 – 0xFF  :  0x00 – 0xFF |

* **sub-function = sendKey**

|  |  |  |
| --- | --- | --- |
| **Data byte** | **Parameter Name** | **Byte Value** |
| #1 | SecurityAcces Request SID | 0x27 |
| #2 | sub-function = [ securityAccessType = sendKey ] | 0x02, 0x04,  0x06,  0x08 – 0x7E |
| #3  :  #n | securityKey[] = [  key#1 (high byte)  :  key#m (low byte) ] | 0x00 – 0xFF  :  0x00 – 0xFF |

**Note:**

**securityKey (high and low bytes)**

The “Key” parameter in the request message is the value generated by the security algorithm corresponding to a specific “Seed” value.

**securityAccessDataRecord**

This parameter record is user optional to transmit data to a server when requesting the seed information. It can e.g. contain an identification of the client that is verified in the server.

**Response Frame Format:**

There are two types of response frames. That is a **Positive response** and a **Negative response**.

* **Positive response**

|  |  |  |
| --- | --- | --- |
| **Data byte** | **Parameter Name** | **Byte Value** |
| #1 | SecurityAccess Response SID | 0x27 + 40 = 0x67 |
| #2 | sub-function = [ securityAccessType ] | 00-7F |
| #3  :  #n | securitySeed[] = [  seed#1 (high byte)  :  seed#m (low byte) ] | 0x00 – 0xFF  :  0x00 – 0xFF |

**Note:**

**securityAccessType**

This parameter is an echo of bits 6 - 0 of the sub-function parameter from the request message.

**securitySeed (high and low bytes)**

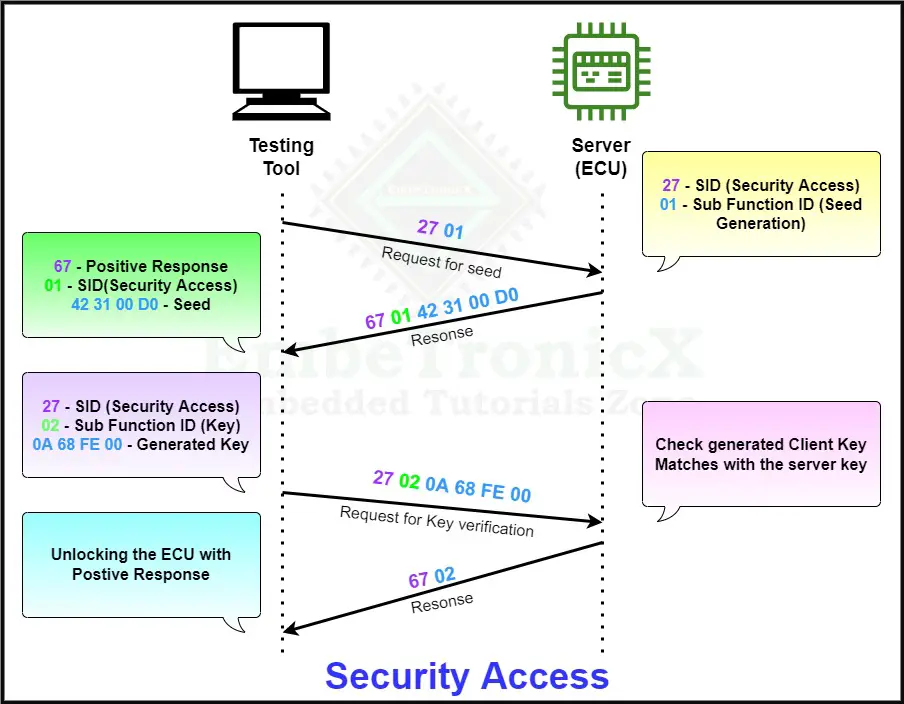
The seed parameter is a data value sent by the server and is used by the client when calculating the key needed to access security. The securitySeed data bytes are only present in the response message if the request message was sent with the sub-function set to a value which requests the seed of the server.

* **Negative response**

The following negative response codes shall be implemented for this service

|  |  |  |  |
| --- | --- | --- | --- |
| **Response ID** | **SID** | **NRC** | **Description** |
| 7F | 27 | 0x12 | **sub-functionNotSupported**  This NRC shall be sent if the sub-function parameter is not supported. |
| 7F | 27 | 0x13 | **incorrectMessageLengthOrInvalidFormat**  This NRC shall be sent if the length of the message is wrong. |
| 7F | 27 | 0x22 | **conditionsNotCorrect**  This NRC shall be returned if the criteria for the request DiagnosticSessionControl are not met. |
| 7F | 27 | 0x24 | **requestSequenceError** RSE  Send if the ‘sendKey’ sub-function is received without first receiving a ‘requestSeed’ request  message. |
| 7F | 27 | 0x31 | **requestOutOfRange**  This NRC shall be sent if the user optional securityAccessDataRecord contains invalid data. |
| 7F | 27 | 0x35 | **invalid Key**  Send if an expected 'sendKey' sub-function value is received and the value of the key does  not match the server's internally stored/calculated key. |
| 7F | 27 | 0x36 | **exceededNumberOfAttempts**  Send if the delay timer is active due to exceeding the maximum number of allowed false  access attempts. |
| 7F | 27 | 0x37 | **requiredTimeDelayNotExpired**  Send if the delay timer is active and a request is transmitted. |

**Example:**



#### **Communication Control (0x28)**

* The communication control’s service ID is 0x28 and the Response SID is 0x68.
* This communication control service is used to control the communication of the ECU server. This means this service enables or disables the transmission or reception of a message from the server(ECU) via a diagnostics line.
* This service is used to control the Transmitter and receiver of the ECU. Testers can use multiple types of communication at the same time.
* This service is quite useful in the context in which the communication wants to be stopped for all or a group of ECUs. For example, this can be used during a software download in order to maximize the available bandwidth.

**There are 3 types of communication types available which are,**

1. Normal communication message (0x01)
2. Network management communication message (0x02)
3. Both Normal communication messages and Network management communication messages (0x03)

**The sub-functions are explained in the below table.**

|  |  |
| --- | --- |
| **Sub-function** | **Description** |
| enableRxAndTx (0x00) | Transmission and Reception of the messages are enabled in this type. |
| enableRxAndDisableTx (0x01) | Transmission of the message is disabled, and the Reception of the message is enabled in this type. |
| disableRxAndEnableTx (0x02) | Transmission of the message is enabled, and the Reception of the message is disabled in this type. |
| disableRxAndTx (0x03) | Both Transmission and Reception of message is disabled in this type of communication. |

**Request Frame Format:**

|  |  |  |
| --- | --- | --- |
| **Data byte** | **Parameter Name** | **Byte Value** |
| #1 | CommunicationControl Request SID | 0x28 |
| #2 | sub-function = [ controlType ] | 0x00 – 0x7F |
| #3 | communicationType | 0x01 – 0x03 |

**Response Frame Format:**

There are two types of response frames. That is a **Positive response** and a **Negative response**.

* **Positive response**

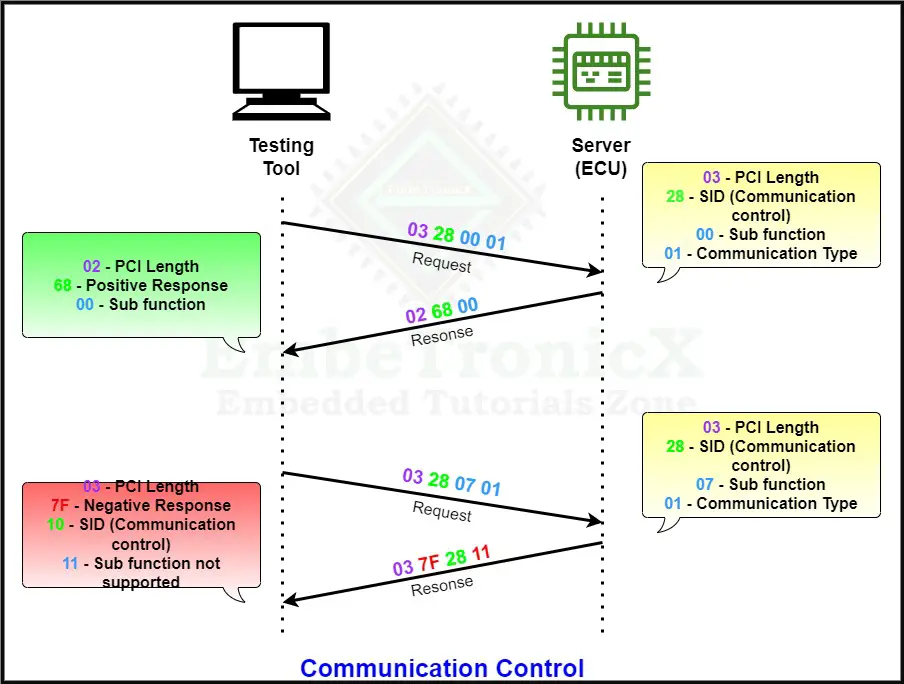
|  |  |  |
| --- | --- | --- |
| **Data byte** | **Parameter Name** | **Byte Value** |
| #1 | CommunicationControl Response SID | 0x28 + 40 = 0x68 |
| #2 | ControlType | 0x00 – 0x7F |

* **Negative response**

The following negative response codes shall be implemented for this service

|  |  |  |  |
| --- | --- | --- | --- |
| **Response ID** | **SID** | **NRC** | **Description** |
| 7F | 28 | 0x12 | **sub-functionNotSupported**  This NRC shall be sent if the sub-function parameter is not supported. |
| 7F | 28 | 0x13 | **incorrectMessageLengthOrInvalidFormat**  This NRC shall be sent if the length of the message is wrong. |
| 7F | 28 | 0x22 | **conditionsNotCorrect**  Used when the server is in a critical normal mode activity and therefore cannot disable/enable the requested communication type. |
| 7F | 28 | 0x31 | **requestOutOfRange**  The server shall use this response code, if it detects an error in the communicationType or nodeIdentificationNumber parameter. |

**Example:**

****

#### **Tester Present (0x3E)**

* The Tester present’s service ID is 0x3E and the Response SID is 0x7E.
* It is one of the important services in UDS protocol, We will see the purpose of this service ID.
* This service is used to indicate to the server, that the client is still connected to the server. This service is used to keep the server alive in the current session except for the default session.
* To keep the client in the current session ( Non-default session) tester present (0x3E) is used and the client will send this service (0x3E) periodically.
* If the client is not exchanging any data with the server for more than the S3 timer value of approximately 5 seconds (this timing value is explained below in the S3 timer), the server (ECU) will get disconnected for safety purposes and automatically fall back to the Default session.

**What is an S3 timer?**

The S3 server parameter is the server (ECU) side timing parameter implemented in each ECUs. The main function of the S3 server parameter is to auto-return into the default session from the non-default session (programming, Extended, Safety system session) after the timeout. The timeout value is based on the S3 parameter only.

The S3 timer will start when the tester requests to change to a non-default session and the S3 timer value may vary depending on OEMs but most of the OEMs are keeping the S3 value as 5 seconds.

**Why do we need of S3 timer?**

If the client is not doing anything in the non-default session for the period, then ECU will automatically go to the default session for safety.

Suppose the server is not going to the default session and staying in the same non-default session after the timeout. Without knowing that If we start the vehicle, ECU will not work. Because of this, accidents or other failures may happen.

**SPRMIB:**

**Suppress Positive Response Message Indication Bit** (SPRMIB) is only supported in sub-function services with sub-function bytes. This bit is used to define whether the positive response of ECU wants to expose or not to the client.

The suppressPosResMsgIndicationBit is supported via the CAN interface only, not on the K-line and ITS interface.

* If the service supports suppressPosResMsgIndicationBit, then bit 7 is interpreted as suppressPosResMsgIndicationBit.
* When bit 7 is ‘1’(SPRMIB=1) then, ECU will not send the positive response to the client which means it will Suppress the positive response.
* When bit 7 is ‘0’ (SPRMIB=0) then, ECU will send the positive response to the client which means No suppression of the positive response.

If the service is not supporting SPRMIB, then it will send the negative response (Subfunction not Supported) with suppressPosResMsgIndicationBit(SPRMIB)=1.

**Note: With SPRMIB we can suppress only positive responses. Negative responses cannot be suppressed.**

**Here are some of the services with sub-functions that support SPRMIB.**

0x10,0x11,0x19,0x27,0x28,0x31,0x3E,0x83,0x85,0x87.

**The sub-functions are explained in the below table.**

|  |  |
| --- | --- |
| **Sub-function** | **Description** |
| zeroSubFunction (0x00) | No sub-function value beside SuppressPosResMsgIndication (SPRMI) not supported |
| (0x80) | SuppressPosResMsgIndication (SPRMI) support (7th bit has been set – (SPRMIB)) |

**Request Frame Format:**

|  |  |  |
| --- | --- | --- |
| **Data byte** | **Parameter Name** | **Byte Value** |
| #1 | TesterPresent Request SID | 0x3E |
| #2 | zeroSubFunction, suppressPosRspMsgIndicationBit = FALSE/ TRUE | 0x00 / 0x80 |

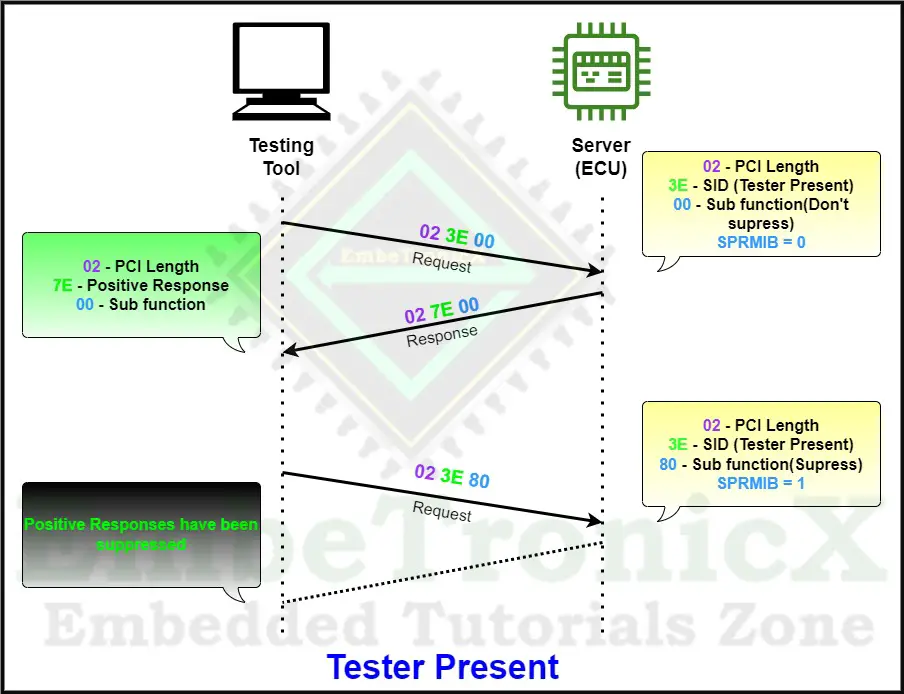
**Response Frame Format:**

There are two types of response frames. That is a **Positive response** and a **Negative response**.

* **Positive response**

|  |  |  |
| --- | --- | --- |
| **Data byte** | **Parameter Name** | **Byte Value** |
| #1 | TesterPresent Response SID | 0x3E + 40 = 0x7E |
| #2 | zeroSubFunction, suppressPosRspMsgIndicationBit = FALSE/ TRUE | 0x00 / 0x80 |

**Example:**

****

* **Negative response**

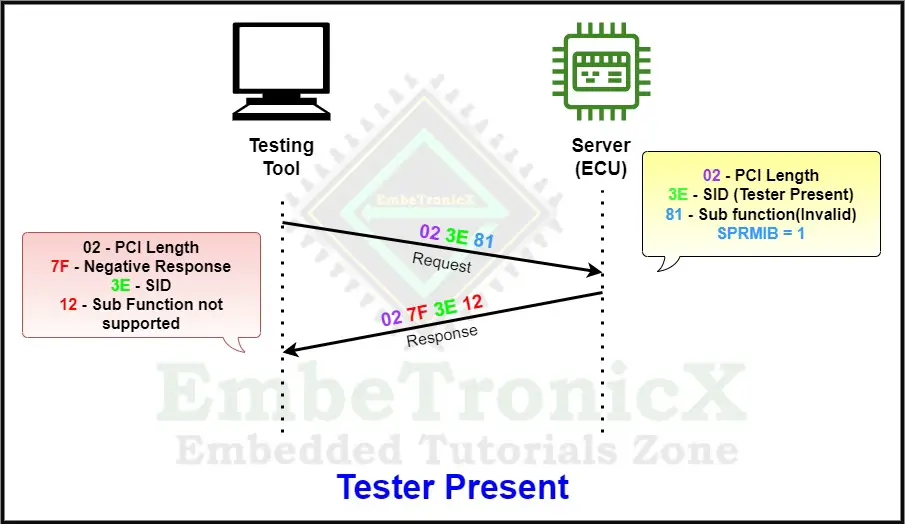
The following negative response codes shall be implemented for this service

If the service is not supporting SPRMIB, then it will send the negative response (Subfunction not Supported) with suppressPosResMsgIndicationBit (SPRMIB) = 1.

We know the Tester present has only two sub-functions which are 0x00 and 0x80. So, 0x01 to 0x7F is not supported. If you send the not-supported sub-function, then it will send a negative response.

|  |  |  |  |
| --- | --- | --- | --- |
| **Response ID** | **SID** | **NRC** | **Description** |
| 7F | 3E | 0x12 | **sub-functionNotSupported**  This NRC shall be sent if the sub-function parameter is not supported. |
| 7F | 3E | 0x13 | **incorrectMessageLengthOrInvalidFormat**  This NRC shall be sent if the length of the message is wrong. |

**Example:**

****

#### **Access Timing Parameter (0x83)**

* The Access timing parameter’s service ID is 0x83 and the Response SID is 0xC3.
* This Access timing parameter service is used to read and write the default timing parameters of a communication link for the duration of this communication. In the communication between the controllers and the client, certain times must be observed.
* If these are exceeded without a single message sent, it must be assumed that the connection was interrupted. By using this service, the Timeout values and message separation time can be read/written.

**The sub-functions are explained in the below table.**

|  |  |
| --- | --- |
| **Sub-function** | **Description** |
| 0x01 | Read Extended Timing Parameter Set |
| 0x02 | Set Timing Parameters To Default Values |
| 0x03 | Read Currently Active Timing Parameters |
| 0x04 | Set Timing Parameters To Given Values |

**Request Frame Format:**

|  |  |  |
| --- | --- | --- |
| **Data byte** | **Parameter Name** | **Byte Value** |
| #1 | AccessTimingParameter Request SID | 0x83 |
| #2 | sub-function = [ timingParameterAccessType ] | 0x00 – 0xFF |
| #3  :  #n | TimingParameterRequestRecord [  byte#1  :  byte#m ] | 0x00 – 0xFF  :  0x00 – 0xFF |

**Response Frame Format:**

There are two types of response frames. That is a **Positive response** and a **Negative response**.

* **Positive response**

|  |  |  |
| --- | --- | --- |
| **Data byte** | **Parameter Name** | **Byte Value** |
| #1 | AccessTimingParameter Response SID | 0x83 + 40 = 0xC3 |
| #2 | timingParameterAccessType | 0x00 – 0x7F |
| #3  :  #n | TimingParameterRequestRecord [  byte#1  :  byte#m ] | 0x00 – 0xFF  :  0x00 – 0xFF |

* **Negative response**

The following negative response codes shall be implemented for this service

|  |  |  |  |
| --- | --- | --- | --- |
| **Response ID** | **SID** | **NRC** | **Description** |
| 7F | 83 | 0x12 | **sub-functionNotSupported**  This NRC shall be sent if the sub-function parameter is not supported. |
| 7F | 83 | 0x13 | **incorrectMessageLengthOrInvalidFormat**  This NRC shall be sent if the length of the message is wrong. |
| 7F | 83 | 0x22 | **conditionsNotCorrect**  This NRC shall be returned if the criteria for the request AccessTimingParameter are not met. |
| 7F | 83 | 0x31 | **requestOutOfRange**  This NRC shall be sent if the TimingParameterRequestRecord contains invalid timing parameter values. |

#### **Secure Data Transmission (0x84)**

The purpose of this service is to transmit data protected against third-party attacks that could jeopardize data security, as per ISO 15764. This service is applicable if a client-server intends to use diagnostic services defined in a secured mode. A secured mode in this context means that the data transmitted is protected by cryptographic methods. The security sub-layer of the transmitter encodes the encapsulated service. The security Sub Layer of the receiver decodes the encapsulated service.

**Request Frame Format:**

|  |  |  |
| --- | --- | --- |
| **Data byte** | **Parameter Name** | **Byte Value** |
| #1 | SecuredDataTransmission Request SID | 0x84 |
| #2  :  #n | securityDataRequestRecord[] = [  securityDataParameter#1  :  securityDataParameter#m ] | 0x00 – 0xFF  :  0x00 – 0xFF |

**Response Frame Format:**

There are two types of response frames. That is a **Positive response** and a **Negative response**.

* **Positive response**

|  |  |  |
| --- | --- | --- |
| **Data byte** | **Parameter Name** | **Byte Value** |
| #1 | SecuredDataTransmission Response SID | 0x84 + 40 = 0xC4 |
| #2  :  #n | securityDataRequestRecord[] = [  securityDataParameter#1  :  securityDataParameter#m ] | 0x00 – 0xFF  :  0x00 – 0xFF |

* **Negative response**

The following negative response codes shall be implemented for this service

|  |  |  |  |
| --- | --- | --- | --- |
| **Response ID** | **SID** | **NRC** | **Description** |
| 7F | 84 | 0x13 | **incorrectMessageLengthOrInvalidFormat** IMLOIF  The server shall use this response code, if the length of the request A\_PDU is not correct. |
| 7F | 84 | 0x38 – 0x4F | **reservedByExtendedDataLinkSecurityDocument**  This range of values is reserved by extended data link security. |

#### **Control DTC setting (0x85)**

* The Control DTC Settings’s service ID is 0x85 and the Response SID is 0xC5.
* Generally, the Diagnostic Trouble Code(DTC) is used to indicate the failure that was caused by the ECU. In some situations, the tester may think not to detect some errors that occurred in the ECU.
* At that time we can use this service. The control DTC Setting service is used when the client/Tester needs to stop or resume the server/ECU updating the DTC when a fault occurs.
* This would be very helpful when the client knows the issue in the vehicle but is trying to find the root cause by doing some additional experiments.
* The control DTC Setting service is used to turn ON or turn OFF the detection of fault error code.
* When the client requests the Server to turn OFF, the Server receives the request and temporarily turns off this feature and the server holds the current value and suspends the new value.
* Later, When the client requests the server to turn ON, then the server will update with the new value.
* If there is a Clear Diagnostic information service (0x14) requested by the user when the control DTC setting is in the OFF state, the server may still clear the DTC information.
* If the client requests the service to be turned OFF when the DTC setting of the server is already in OFF mode, the server responds with a positive response and it will ignore the request.

**The sub-functions are explained in the below table.**

|  |  |
| --- | --- |
| **Sub-function** | **Description** |
| 0x01 | **ON**  The server(s) shall resume the updating of diagnostic trouble code status bits according to normal operating conditions |
| 0x02 | **OFF**  The server(s) shall stop the updating of diagnostic trouble code status bits. |

**Request Frame Format:**

|  |  |  |
| --- | --- | --- |
| **Data byte** | **Parameter Name** | **Byte Value** |
| #1 | ControlDTCSetting Request SID | 0x85 |
| #2 | DTCSettingType = off/on, suppressPosRspMsgIndicationBit = FALSE | 0x01 / 0x02 |

**Response Frame Format:**

There are two types of response frames. That is a **Positive response** and a **Negative response**.

* **Positive response**

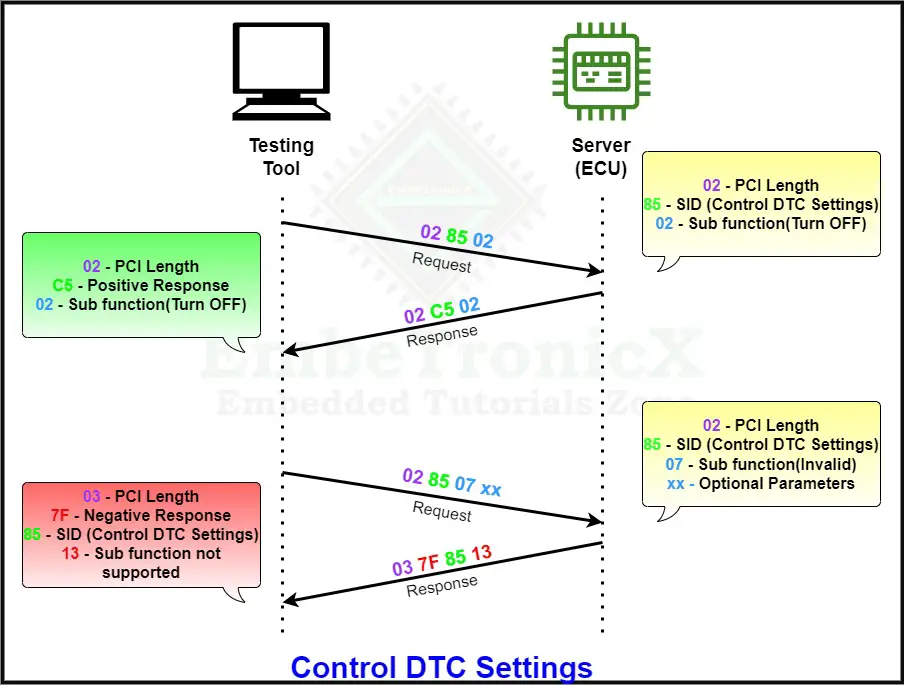
|  |  |  |
| --- | --- | --- |
| **Data byte** | **Parameter Name** | **Byte Value** |
| #1 | ControlDTCSetting Response SID | 0x85 + 40 = 0xC5 |
| #2 | DTCSettingType = off/on | 0x01 / 0x02 |

* **Negative response**

The following negative response codes shall be implemented for this service

|  |  |  |  |
| --- | --- | --- | --- |
| **Response ID** | **SID** | **NRC** | **Description** |
| 7F | 85 | 0x12 | **sub-functionNotSupported**  This NRC shall be sent if the sub-function parameter is not supported. |
| 7F | 85 | 0x13 | **incorrectMessageLengthOrInvalidFormat**  This NRC shall be sent if the length of the message is wrong. |
| 7F | 85 | 0x22 | **conditionsNotCorrect**  Used when the server is in a critical normal mode activity and therefore cannot perform the requested DTC control functionality. |
| 7F | 85 | 0x31 | **requestOutOfRange**  The server shall use this response code, if it detects an error in the DTCSettingControlOptionRecord. |

**Example:**

****

#### **Response To Event (0x86)**

* The Response On Event’s service ID is 0x86 and the Response SID is 0xC6.
* This Response On Event service requests the server to start or stop the transmission of the response on a specific event. This service automatically executes the diagnostic service when a specific event occurs without getting a request from the client.
* The client specifies the event with optional event parameters and the service with service parameters to be executed if the event occurs. Using this service tester can Configure the ECU to send a response to the tester without a request in case any defined event occurs.
* Let’s say, for example, Power interruption is an event in a vehicle. If a power interruption occurs while driving the ECU, it will not wait for the tester’s request. The ECU will send the response to the tester/client. In this case, the tester will receive a negative response.

**The sub-functions are explained in the below table.**

|  |  |
| --- | --- |
| **Sub-function** | **Description** |
| 0x00 | Stop Response On Event |
| 0x01 | On DTC Status Change |
| 0x02 | On Timer Interrupt |
| 0x03 | On Change Of Data Identifier |
| 0x04 | Report Activated Events |
| 0x05 | Start Response On Event |
| 0x06 | Clear Response On Event |
| 0x07 | On Comparison Of Values |

**Request Frame Format:**

|  |  |  |
| --- | --- | --- |
| **Data byte** | **Parameter Name** | **Byte Value** |
| #1 | ResponseOnEvent Request SID | 0x86 |
| #2 | sub-function = [ eventType ] | 0x00 - 0x07 |

**Response Frame Format:**

There are two types of response frames. That is a **Positive response** and a **Negative response**.

* **Positive response**

|  |  |  |
| --- | --- | --- |
| **Data byte** | **Parameter Name** | **Byte Value** |
| #1 | ResponseOnEvent Response SID | 0x86 + 40 = 0xC6 |
| #2 | sub-function = [ eventType ] | 0x00 - 0x07 |

* **Negative response**

The following negative response codes shall be implemented for this service

|  |  |  |  |
| --- | --- | --- | --- |
| **Response ID** | **SID** | **NRC** | **Description** |
| 7F | 86 | 0x12 | **sub-functionNotSupported**  This NRC shall be sent if the sub-function parameter is not supported. |
| 7F | 86 | 0x13 | **incorrectMessageLengthOrInvalidFormat**  This NRC shall be sent if the length of the message is wrong. |
| 7F | 86 | 0x22 | **conditionsNotCorrect**  Used when the server is in a critical normal mode activity and therefore cannot perform the requested functionality. |
| 7F | 86 | 0x31 | **requestOutOfRange**  The server shall use this response code  if it detects an error in the eventTypeRecord parameter;  if the specified eventWindowTime is invalid;  if the requested DID is not supported;  if a combination of finite event window and storageState equal to storeEvent is requested |

#### **Link Control (0x87)**

* The Link control ‘s service ID is 0x87 and the Response SID is 0xC7.
* Link control can be used to change the communication parameters between the client and server to increase the baud rate. So, the tester can exchange the data between the Tester and the server very quickly.

**The sub-functions are explained in the below table.**

|  |  |
| --- | --- |
| **Sub-function** | **Description** |
| 0x01 | **verifyModeTransitionWithFixedParameter**  This parameter is used to verify if a transition with a pre-defined parameter, which is specified by the linkControlModeIdentifier data-parameter can be performed. |
| 0x02 | **verifyModeTransitionWithSpecificParameter**  This parameter is used to verify if a transition to a specifically defined  parameter (e.g., specific baudrate), which is specified by the linkRecord  data-parameter can be performed. |
| 0x03 | **transitionMode**  This sub-function parameter requests the server(s) to transition the data link into the mode which was requested in the preceding verification message. |

**Request Frame Format:**

|  |  |  |
| --- | --- | --- |
| **Data byte** | **Parameter Name** | **Byte Value** |
| #1 | LinkControl Request SID | 0x87 |
| #2 | sub-function = [ linkControlType ] | 0x01 - 0x03 |

**Response Frame Format:**

There are two types of response frames. That is a **Positive response** and a **Negative response**.

* **Positive response**

|  |  |  |
| --- | --- | --- |
| **Data byte** | **Parameter Name** | **Byte Value** |
| #1 | LinkControl Response SID | 0x87 + 40 = 0xC7 |
| #2 | inkControlType | 0x01 - 0x03 |

* **Negative response**

The following negative response codes shall be implemented for this service

|  |  |  |  |
| --- | --- | --- | --- |
| **Response ID** | **SID** | **NRC** | **Description** |
| 7F | 87 | 0x12 | **sub-functionNotSupported**  This NRC shall be sent if the sub-function parameter is not supported. |
| 7F | 87 | 0x13 | **incorrectMessageLengthOrInvalidFormat**  This NRC shall be sent if the length of the message is wrong. |
| 7F | 87 | 0x22 | **conditionsNotCorrect**  This NRC shall be returned if the criteria for the requested LinkControl are not met. |
| 7F | 87 | 0x24 | **requestSequenceError**  This NRC shall be returned if the client requests the transition of the mode of operation without a preceding verification step, which specifies the mode to transition to. |
| 7F | 87 | 0x31 | **requestOutOfRange**  This NRC shall be returned if  the requested linkControlModeIdentifier is invalid;  the specific modeParameter (linkRecord) is invalid; |

### **Data Transmission**

Normally, data transmission is the transfer of data from one device to another device. The data transfer will use many communication methods like I2C, SPI, etc. In those communication protocols, we will have to follow some specific procedure to read/write data from/to the memory or device. Like that, in the UDS protocol, we will have to follow some procedure to read or write data. We will get into the Data Transmission in UDS Protocol.

Data Transmission in UDS Protocol is one type of functional group that has the following services used for data transmission between the server(ECU) and client(Tester). These are some important DIDs that we use very often.

1. **Read Data by Identifier (0x22)**
2. **Read Memory by Address (0x23)**
3. **Read Scaling data by Identifier (0x24)**
4. **Read Data by Periodic Identifier (0x2A)**
5. **Dynamically define data Identifier (0x2C)**
6. **Write Data by Identifier (0x2E)**
7. **Write Memory by Address (0x3D)**

Before going into detail about each service, we will learn what is Data by Identifier (DID) first.

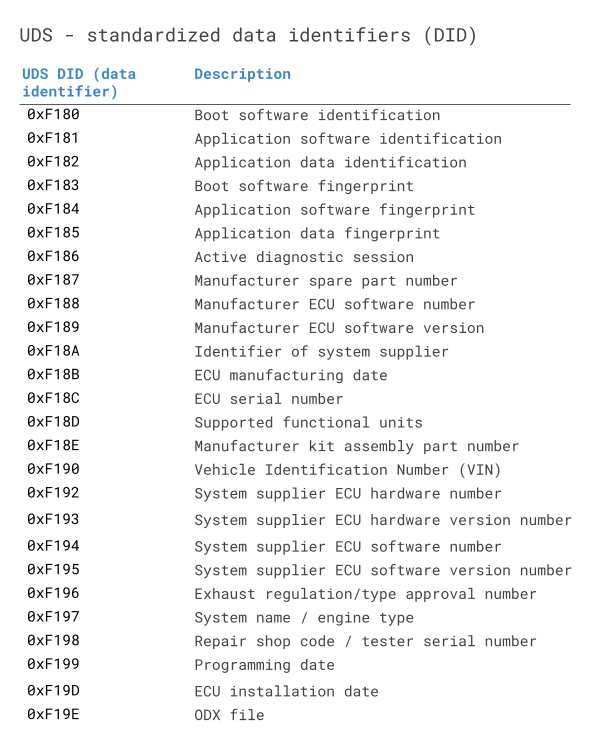
**Data by Identifier (DID)**

Data by Identifier is a 2-byte identification code. DID is an important component for generating ECU diagnostics and troubleshooting a malfunctioning ECU. This 16-bit ID (0-65535) combines a piece of data, its length, and its description together in a DID. What each DID stands for is entirely within the scope of the computer designer.

There are many diagnostics services using DIDs. Read data byte identifier and Write data byte identifier are the most popular services using DIDs.

**Some of the Example DIDs are mentioned below:**

The DID may vary depending on the OEMs. some of them may use different DIDs also.

****

#### **Read Data by Identifier (0x22)**

* Read Data by Identifier SID is 0x22 and the Response is 0x62.
* Read Data by Identifier services allows the tester to request the server for recorded data. With this service, the tester can retrieve one or more data from the server.
* To read some sensitive data from Server using Read Data by Identifiers, the identifiers may require security access granted from the server. After getting security access from the server, then only the Client can read the data. The data identifiers list should be present with the client to read the data, else there would be a negative response sent by the server. The client can request more than one data by identifier at the same time. The ECU would combine the data for all the DIDs in one single positive response to the client.
* The tester can Read the Software version of the ECU, Vehicle Identification Number, position of the vehicle, sensor data, etc., Each value is associated with a 16-bit unique DID.

**Request Frame Format:**

|  |  |  |
| --- | --- | --- |
| **Data byte** | **Parameter Name** | **Byte Value** |
| #1 | ReadDataByIdentifier Request SID | 0x22 |
| #2  #3 | dataIdentifier[]#1 = [  byte#1 (MSB)  byte#2 ] | 0x00 – 0xFF  0x00 – 0xFF |
| #n-1  #n | dataIdentifier[]#m = [  byte#1 (MSB)  byte#2 ] | 0x00 – 0xFF  0x00 – 0xFF |

**Response Frame Format:**

There are two types of response frames. That is a **Positive response** and a **Negative response**.

* **Positive response**

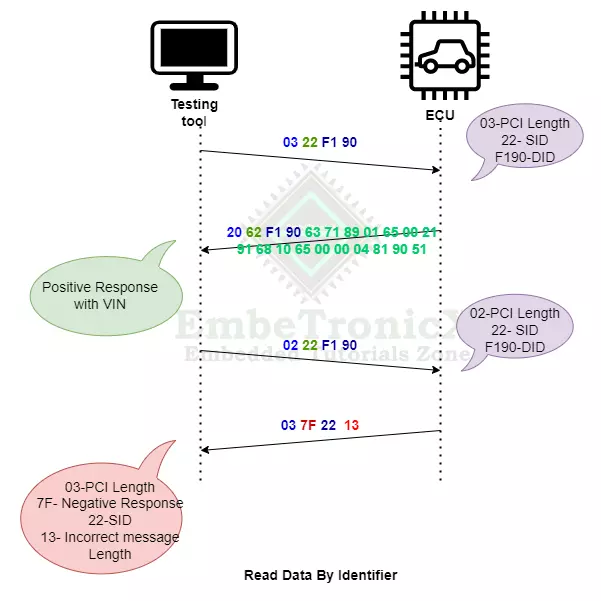
|  |  |  |
| --- | --- | --- |
| **Data byte** | **Parameter Name** | **Byte Value** |
| #1 | ReadDataByIdentifier Response SID | 0x22 + 40 = 0x62 |
| #2  #3 | dataIdentifier[]#1 = [  byte#1 (MSB)  byte#2 ] | 0x00 – 0xFF  0x00 – 0xFF |
| #n-1  #n | dataIdentifier[]#m = [  byte#1 (MSB)  byte#2 ] | 0x00 – 0xFF  0x00 – 0xFF |

* **Negative response**

The following negative response codes shall be implemented for this service

|  |  |  |  |
| --- | --- | --- | --- |
| **Response ID** | **SID** | **NRC** | **Description** |
| 7F | 22 | 0x13 | **incorrectMessageLengthOrInvalidFormat**  This NRC shall be sent if the length of the request message is invalid or the client exceeded  the maximum number of dataIdentifiers allowed to be requested at a time. |
| 7F | 22 | 0x14 | **responseTooLong**  This NRC shall be sent if the total length of the response message exceeds the limit of the  underlying transport protocol (e.g., when multiple DIDs are requested in a single request). |
| 7F | 22 | 0x22 | **conditionsNotCorrect**  This NRC shall be sent if the operating conditions of the server are not met to perform the  required action. |
| 7F | 22 | 0x31 | **requestOutOfRange**  This NRC shall be sent if  none of the requested dataIdentifier values are supported by the device;  none of the requested dataIdentifiers are supported in the current session;  the requested dynamicDefinedDataIdentifier has not been assigned yet; |
| 7F | 22 | 0x33 | **securityAccessDenied**  This NRC shall be sent if at least one of the dataIdentifiers is secured and the server is not in  an unlocked state. |

**Example:**



#### **Read Memory by Address (0x23)**

* Read memory by address SID is 0x23 and the Response is 0x63.
* Read memory by address services allow the diagnostic tool to read the information from a certain region of the memory in ECU. The client requests information from the ECU by providing the starting memory address and the size of the memory to be read.
* The starting memory address and memory size to be read is mentioned in the address and Length Format Identifier (low and high nibble) section of the request message. It is a 1- byte parameter.
* A lower nibble (bit 0 to bit 3 ) indicates the memory address parameter and a Higher nibble(bit 4 to bit 7) indicates the memory size parameter.
* For example, assume the address and length format identifier value is 0x24. Upper nibble 0x2 indicates memory address and lower nibble 0x4 indicates memory size. We can use a fixed address, Length Format Identifier, and unused bytes within the memory Address or memory Size parameter are padded with the value 0x00 in the higher range address locations.
* While trying to read some specific memory region, ECU will ask the client whether the SID has security access to read the data or not. Once security access is granted by the ECU then only the diagnostic tool can able to read the data from the memory.

**Request Frame Format:**

|  |  |  |
| --- | --- | --- |
| **Data byte** | **Parameter Name** | **Byte Value** |
| #1 | ReadMemoryByAddress Request SID | 0x23 |
| #2 | addressAndLengthFormatIdentifier | 0x00 – 0xFF |
| #3  :  #(m-1)+3 | memoryAddress[] = [  byte#1 (MSB)  :  byte#m ] | 0x00 – 0xFF  0x00 – 0xFF |
| #n-(k-1)  :  #n | memorySize[] = [  byte#1 (MSB)  :  byte#k ] | 0x00 – 0xFF  0x00 – 0xFF |

**Response Frame Format:**

There are two types of response frames. That is a **Positive response** and a **Negative response**.

* **Positive response**

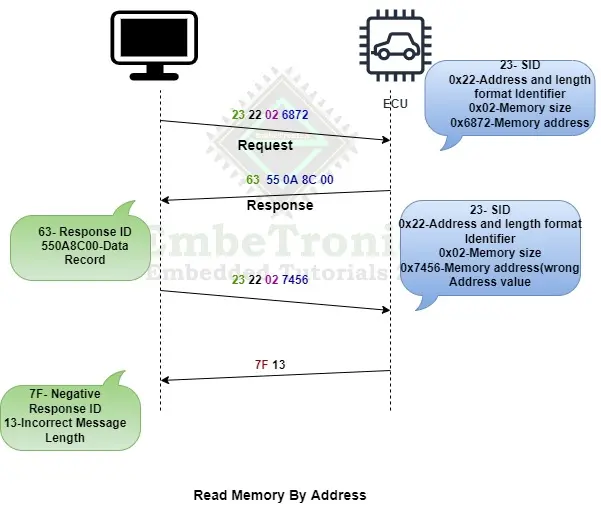
|  |  |  |
| --- | --- | --- |
| **Data byte** | **Parameter Name** | **Byte Value** |
| #1 | ReadMemoryByAddress Response SID | 0x23 + 40 = 0x63 |
| #2  :  #n | dataRecord[] = [  data#1  :  data#m ] | 0x00 – 0xFF  :  0x00 – 0xFF |

* **Negative response**

The following negative response codes shall be implemented for this service

|  |  |  |  |
| --- | --- | --- | --- |
| **Response ID** | **SID** | **NRC** | **Description** |
| 7F | 23 | 0x13 | **incorrectMessageLengthOrInvalidFormat**  This NRC shall be sent if the length of the message is wrong. |
| 7F | 23 | 0x22 | **conditionsNotCorrect**  This NRC shall be sent if the operating conditions of the server are not met to perform the  required action. |
| 7F | 23 | 0x31 | **requestOutOfRange**  This NRC shall be sent if:   * Any memory address within the interval [0xMA, (0xMA + 0xMS -0x1)] is invalid; * Any memory address within the interval [0xMA, (0xMA + 0xMS -0x1)] is restricted; * The memorySize parameter value in the request message is not supported by the * server; * The specified addressAndLengthFormatIdentifier is not valid; * The memorySize parameter value in the request message is zero; |
| 7F | 23 | 0x33 | **SecurityAccessDenied**  This NRC shall be sent if any memory address within the interval [0xMA, (0xMA + 0xMS -  0x1)] is secure and the server is locked. |

**Example:**



#### **Read Scaling data by Identifier (0x24)**

* Scaling data means, a set of values that can be attributed to a formula can be read by the client. For example, We have to convert the data to a digital signal when we read the analog signal from the sensor. In order to change it to its equivalent digital value, we will use some constant value. Those values are called scaling data.
* Read Scaling Data by Identifier SID is 0x24 and the Response is 0x64.
* This service is used to read the scaling information of the record identified by a provided data identifier.

**Request Frame Format:**

|  |  |  |
| --- | --- | --- |
| **Data byte** | **Parameter Name** | **Byte Value** |
| #1 | ReadScalingDataByIdentifier Request SID | 0x24 |
| #2  #3 | dataIdentifier [ byte#1 ] (MSB)  dataIdentifier [ byte#2 ] (LSB) | 0x00 – 0xFF  0x00 – 0xFF |

**Response Frame Format:**

There are two types of response frames. That is a **Positive response** and a **Negative response**.

* **Positive response**

|  |  |  |
| --- | --- | --- |
| **Data byte** | **Parameter Name** | **Byte Value** |
| #1 | ReadScalingDataByIdentifier Response SID | 0x24 + 40 = 0x64 |
| #2  #3 | dataIdentifier [ byte#1 ] (MSB)  dataIdentifier [ byte#2 ] (LSB) | 0x00 – 0xFF  0x00 – 0xFF |
| #4 | scalingByte#1 {ASCII, 15 data bytes} | 0x00 – 0xFF |
| #5 | scalingByte#2 {ASCII, 2 data bytes} | 0x00 – 0xFF |

* **Negative response**

The following negative response codes shall be implemented for this service

|  |  |  |  |
| --- | --- | --- | --- |
| **Response ID** | **SID** | **NRC** | **Description** |
| 7F | 24 | 0x13 | **incorrectMessageLengthOrInvalidFormat**  This NRC shall be sent if the length of the message is wrong. |
| 7F | 24 | 0x22 | **conditionsNotCorrect**  This NRC shall be sent if the operating conditions of the server are not met to perform the  required action. |
| 7F | 24 | 0x31 | **requestOutOfRange**  This NRC shall be returned if:  the requested dataIdentifier value is not supported by the device, the requested dataIdentifier value is supported by the device, but no scaling information is available for the specified dataIdentifier. |
| 7F | 24 | 0x33 | **securityAccessDenied**  This NRC shall be sent if the dataIdentifier is secured and the server is not in an unlocked  state. |

**Example:**

In the below scaling Data, 0x6F 0x62 are scaling data information.

**0x6F** – Indicates,

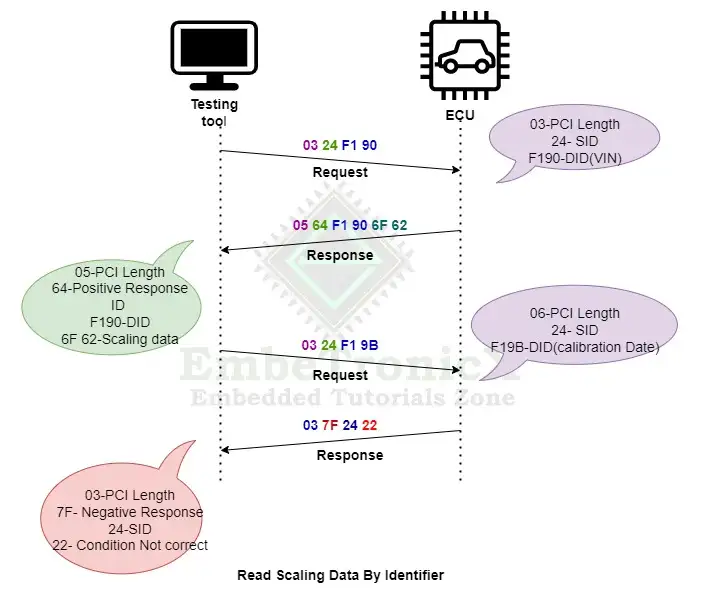
6 – Represents that the value is an ASCII value

F – Represents the total number of bytes, in this case, 15 bytes of VIN

**0x62** – Indicates,

6 – Represents that the value is an ASCII value

2 – Represents the 2 leftover bytes of the VIN



#### **Read Data by Periodic Identifier (0x2A)**

* Read data by periodic Identifier is 0x2A and the response is 0x6A. The Read data by Periodic Identifier service allows the client to request the server to periodically transmit the response for one or more data Identifiers.
* According to a request message from the server, the periodic identifier response transmission mode would be set. A new request message must be sent to the server/ECU if it has to be modified. The server will start the timer for the periodic transmission of data after transmitting a positive response with the data identifier only for the request message. To stop the service, the client must send a request message with the transmission mode set to “Stop“.
* The ECU may limit the number of identifiers that can be supported at the same time based on the vehicle manufacturer’s design requirements. If the number of supported simultaneous identifiers is exceeded, the ECU may respond with a negative response, and none of the periodic identifiers may be scheduled.

**Some supporting transmission modes are below table.**

|  |  |
| --- | --- |
| **Transmission modes** | **Description** |
| 0x00 | Reserved |
| 0x01 | Send at a slow rate |
| 0x02 | Send at a medium rate |
| 0x03 | Send at a fast rate |
| 0x04 | Stop sending |
| 0x05 – 0xFF | Reserved |

**Request Frame Format:**

|  |  |  |
| --- | --- | --- |
| **Data byte** | **Parameter Name** | **Byte Value** |
| #1 | ReadDataByPeriodicIdentifier Request SID | 0x2A |
| #2 | transmissionMode | 0x00 – 0xFF |
| #3 | periodicDataIdentifier[]#1 | 0x00 – 0xFF |
| #n | periodicDataIdentifier[]#n | 0x00 – 0xFF |

**Response Frame Format:**

There are two types of response frames. That is a **Positive response** and a **Negative response**.

* **Positive response**

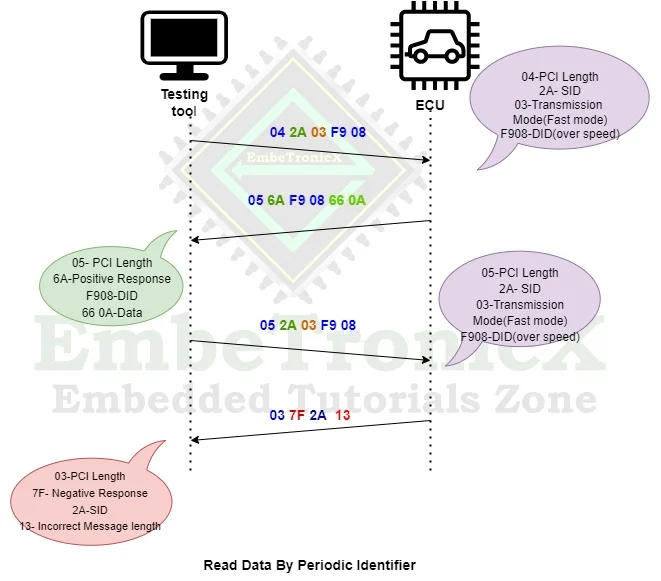
|  |  |  |
| --- | --- | --- |
| **Data byte** | **Parameter Name** | **Byte Value** |
| #1 | ReadDataByPeriodicIdentifier Response SID | 0x2A + 40 = 0x6A |
| #2 | periodicDataIdentifier | 0x00 – 0xFF |
| #3  ..  #n | dataRecord[] = [  data#1  :  data#k ] | 0x00 – 0xFF  :  0x00 – 0xFF |

* **Negative response**

The following negative response codes shall be implemented for this service

|  |  |  |  |
| --- | --- | --- | --- |
| **Response ID** | **SID** | **NRC** | **Description** |
| 7F | 2A | 0x13 | **incorrectMessageLengthOrInvalidFormat** IMLOIF  This NRC shall be sent if the length of the request message is invalid or the client exceeded  the maximum number of periodicDataIdentifiers allowed to be requested at a time. |
| 7F | 2A | 0x22 | **conditionsNotCorrect** CNC  This NRC shall be sent if the operating conditions of the server are not met to perform the  required action. E.g. this could occur if the client requests periodicDataIdentifiers with  different transmissionModes and the server does not support multiple transmissionModes  simultaneously. |
| 7F | 2A | 0x31 | **requestOutOfRange** ROOR  This NRC shall be sent if  none of the requested periodicDataIdentifier values are supported by the device;  none of the requested periodicDataIdentifiers are supported in the current session;  The specified transmissionMode is not supported by the device;  the requested dynamicDefinedDataIdentifier has not been assigned yet;  the client exceeded the maximum number of periodicDataIdentifiers allowed to be scheduled  concurrently |
| 7F | 2A | 0x33 | **securityAccessDenied** SAD  This NRC shall be sent if at least one of the periodicDataIdentifier is secured and the server  is not in an unlocked state. |

**Example:**

****

#### **Dynamically define data Identifier (0x2C)**

* The Dynamically define data Identifier service ID is 0x2C and the response is 0x6C.
* This service is generally recommended to be used during ECU development. This service is used when the client needs to read the response of a list of pre-defined data identifier values with another specific customized identifier. It can create a container with an identifier and add all the pre-defined identifiers to it and when this dynamically created identifier is requested, the server responds with the values of all the added pre-defined identifiers.
* It is not recommended to configure one dynamically defined data identifier in another dynamically defined identifier although it can still be done. The server will maintain the records of the dynamically created data identifiers until it is cleared by the client or as per vehicle manufacturer design.

**This service has sub-functions that have been explained in the below table.**

|  |  |
| --- | --- |
| **sub-functions** | **Description** |
| 0x00 | Reserved |
| 0x01 | Define by Identifier |
| 0x02 | Define by Address |
| 0x04 – 0x7F | Clear Dynamically Defined Data Identifier |

**Request Frame Format:**

|  |  |  |
| --- | --- | --- |
| **Data byte** | **Parameter Name** | **Byte Value** |
| #1 | DynamicallyDefineDataIdentifier Request SID | 0x2C |
| #2 | sub-function | 0x00 – 0x7F |
| #3  #4 | dynamicallyDefinedDataIdentifier[] = [  byte#1 (MSB)  byte#2 (LSB) ] | 0xF2 / 0xF3  0x00 – 0xFF |
| #5  #6 | sourceDataIdentifier[]#1 = [  byte#1 (MSB)  byte#2 (LSB) ] | 0x00 – 0xFF  0x00 – 0xFF |
| #7 | positionInSourceDataRecord#1 | 0x01 – 0xFF |
| #8 | memorySize#1 | 0x00 – 0xFF |

**Response Frame Format:**

There are two types of response frames. That is a **Positive response** and a **Negative response**.

* **Positive response**

|  |  |  |
| --- | --- | --- |
| **Data byte** | **Parameter Name** | **Byte Value** |
| #1 | DynamicallyDefineDataIdentifier Response SID | 0x2C + 40 = 0x6C |
| #2 | sub-function | 0x00 – 0x7F |
| #3  ..  #4 | dynamicallyDefinedDataIdentifier [] = [  byte#1 (MSB)  byte#2 (LSB) ] | 0xF2 / 0xF3  0x00 – 0xFF |

* **Negative response**

The following negative response codes shall be implemented for this service

|  |  |  |  |
| --- | --- | --- | --- |
| **Response ID** | **SID** | **NRC** | **Description** |
| 7F | 2C | 0x12 | **sub-functionNotSupported**  This NRC shall be sent if the sub-function parameter is not supported. |
| 7F | 2C | 0x13 | **incorrectMessageLengthOrInvalidFormat**  This NRC shall be sent if the length of the message is wrong. |
| 7F | 2C | 0x22 | **conditionsNotCorrect**  This NRC shall be sent if the operating conditions of the server are not met to perform the  required action. |
| 7F | 2C | 0x31 | **requestOutOfRange**  This NRC shall be sent if:   * Any data identifier (dynamicallyDefinedDataIdentifier or any sourceDataIdentifier) in the request message is not supported/invalid; * The positionInSourceDataRecord was incorrect (less than 1, or greater than maximum allowed by server);Any memory address in the request message is not supported in the server. * The specified memorySize was invalid; * The amount of data to be packed into the dynamic data identifier exceeds the maximum allowed by the server; * The specified addressAndLengthFormatIdentifier is not valid; * The total length of a dynamically defined periodicDataIdentifier exceeds the maximum length that fits into a single frame of the data link used for transmission of the periodic response message; |
| 7F | 2C | 0x33 | **securityAccessDenied**  This NRC shall be sent if:   * Any data identifier (dynamicallyDefinedDataIdentifier or any sourceDataIdentifier) in the request message is secured and the server is not in an unlocked state; * Any memory address in the request message is secured and the server is not in an unlocked state; |

#### **Write Data by Identifier (0x2E)**

Write Data by Identifier SID is 0x2E and the Response is 0x6E.

Write data by identifier service allows the client (diagnostic tool) to write the information on the ECU at the memory location with the help of a Data Identifier (DID).

**Request Frame Format:**

|  |  |  |
| --- | --- | --- |
| **Data byte** | **Parameter Name** | **Byte Value** |
| #1 | WriteDataByIdentifier Request SID | 0x2E |
| #2  #3 | dataIdentifier[] = [  byte#1 (MSB)  byte#2 ] | 0xF1  0x91 |
| #4  #20 | dataRecord[] = [  data#1  :  data#17 bytes ] | 0x00 – 0xFF  :  0x00 – 0xFF |

**Response Frame Format:**

There are two types of response frames. That is a **Positive response** and a **Negative response**.

* **Positive response**

|  |  |  |
| --- | --- | --- |
| **Data byte** | **Parameter Name** | **Byte Value** |
| #1 | WriteDataByIdentifier Response SID | 0x2E + 40 = 6E |
| #2  #3 | dataIdentifier [ byte#1 ] (MSB)  dataIdentifier [ byte#2 ] (LSB) | 0xF1  0x91 |

* **Negative response**

The following negative response codes shall be implemented for this service

|  |  |  |  |
| --- | --- | --- | --- |
| **Response ID** | **SID** | **NRC** | **Description** |
| 7F | 2E | 0x13 | **incorrectMessageLengthOrInvalidFormat**  This NRC shall be sent if the length of the message is wrong. |
| 7F | 2E | 0x22 | **conditionsNotCorrect**  This NRC shall be sent if the operating conditions of the server are not met to perform the required action. |
| 7F | 2E | 0x31 | **requestOutOfRange**  This NRC shall be sent if:   * The dataIdentifier in the request message is not supported in the server or the dataIdentifier is supported for read only purpose (via ReadDataByIdentifier service); * Any data transmitted in the request message after the dataIdentifier is invalid (if applicable to the node); |
| 7F | 2E | 0x33 | **securityAccessDenied**  This NRC shall be sent if the dataIdentifier, which reference a specific address, is secured and the server is not in an unlocked state. |
| 7F | 2E | 0x72 | **generalProgrammingFailure**  This NRC shall be returned if the server detects an error when writing to a memory location. |

#### **Write Memory by Address (0x3D)**

Write Memory by Address service ID is 0x3D and Response ID is 0x7D.

Write Memory by Address service allows the diagnostic tool to write the information in ECU one or contiguous memory region. This service is used to clear the Non-volatile memory and to change the calibrating values directly in memory.

**Request Frame Format:**

|  |  |  |
| --- | --- | --- |
| **Data byte** | **Parameter Name** | **Byte Value** |
| #1 | WriteMemoryByAddress Request SID | 0x3D |
| #2 | addressAndLengthFormatIdentifier | 0x12 |
| #3  #4 | memoryAddress [ byte#1 ] (MSB)  memoryAddress [ byte#2 ] (LSB) | 0x20  0x48 |
| #5 | memorySize [ byte#1 ] | 0x02 |
| #6  #7 | dataRecord [ data#1 ]  dataRecord [ data#2 ] | 0x00  0x8C |

**Response Frame Format:**

There are two types of response frames. That is a **Positive response** and a **Negative response**.

* **Positive response**

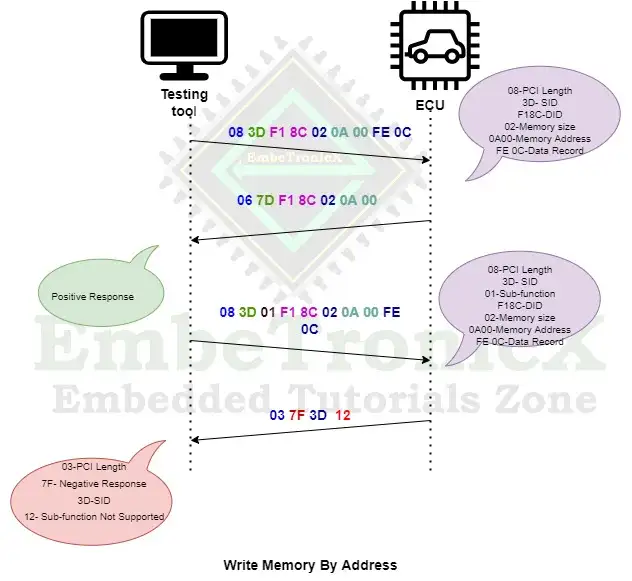
|  |  |  |
| --- | --- | --- |
| **Data byte** | **Parameter Name** | **Byte Value** |
| #1 | WriteMemoryByAddress Response SID | 0x3D + 40 = 7D |
| #2 | addressAndLengthFormatIdentifier | 0x12 |
| #3  #4 | memoryAddress [ byte#1 ] (MSB)  memoryAddress [ byte#2 ] (LSB) | 0x20  0x48 |
| #5 | memorySize [ byte#1 ] | 0x02 |

* **Negative response**

The following negative response codes shall be implemented for this service

|  |  |  |  |
| --- | --- | --- | --- |
| **Response ID** | **SID** | **NRC** | **Description** |
| 7F | 3D | 0x13 | **incorrectMessageLengthOrInvalidFormat**  This NRC shall be sent if the length of the message is wrong. |
| 7F | 3D | 0x22 | **conditionsNotCorrect**  This NRC shall be sent if the operating conditions of the server are not met to perform the required action. |
| 7F | 3D | 0x31 | **requestOutOfRange**  This NRC shall be sent if:   * Any memory address within the interval [0xMA, (0xMA + 0xMS -0x1)] is invalid; * Any memory address within the interval [0xMA, (0xMA + 0xMS -0x1)] is restricted; * The memorySize parameter value in the request message is not supported by the server; * The specified addressAndLengthFormatIdentifier is not valid; * The memorySize parameter value in the request message is zero; |
| 7F | 3D | 0x33 | **securityAccessDenied**  This NRC shall be sent if any memory address within the interval [0xMA, (0xMA + 0xMS -0x1)] is secure and the server is locked. |
| 7F | 3D | 0x72 | **generalProgrammingFailure**  This NRC shall be returned if the server detects an error when writing to a memory location. |

**Example:**



### **Stored Data Transmission**

There are two services are present inside stored data transmission

#### **Clear Diagnostic Information (14 hex)**

The Clear Diagnostic Information service in the UDS Protocol, identified by the code 14 hex, is used to reset or clear diagnostic information stored in the electronic control unit (ECU). When this service is requested, the ECU removes or resets fault codes, diagnostic trouble codes (DTCs), and other diagnostic-related information. It helps in clearing the diagnostic history and allows for a fresh start in monitoring and diagnosing the vehicle’s condition. The Clear Diagnostic Information service plays a crucial role in maintenance and troubleshooting processes, ensuring accurate and up-to-date information for effective vehicle diagnostics.

**Request Frame Format:**

|  |  |  |
| --- | --- | --- |
| **Data byte** | **Parameter Name** | **Byte Value** |
| #1 | ClearDiagnosticInformation Request SID | 0x14 |
| #2  #3  #4 | groupOfDTC [ DTCHighByte ] (“Emissions-related systems”)  groupOfDTC [ DTCMiddleByte ]  groupOfDTC [ DTCLowByte ] | 0xFF  0xFF  0x33 |

**Response Frame Format:**

There are two types of response frames. That is a **Positive response** and a **Negative response**.

* **Positive response**

|  |  |  |
| --- | --- | --- |
| **Data byte** | **Parameter Name** | **Byte Value** |
| #1 | ClearDiagnosticInformation Response SID | 0x14 + 40 = 54 |

* **Negative response**

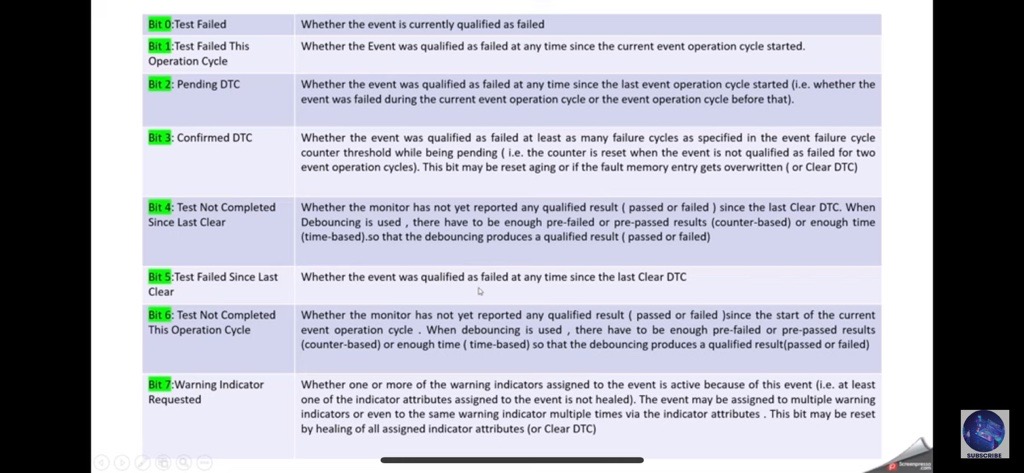
The following negative response codes shall be implemented for this service

|  |  |  |  |
| --- | --- | --- | --- |
| **Response ID** | **SID** | **NRC** | **Description** |
| 7F | 14 | 0x13 | **incorrectMessageLengthOrInvalidFormat**  This NRC shall be sent if the length of the message is wrong. |
| 7F | 14 | 0x22 | **conditionsNotCorrect**  This NRC shall be sent if the operating conditions of the server are not met to perform the required action. |
| 7F | 14 | 0x31 | **requestOutOfRange**  This NRC shall be returned if the specified groupOfDTC parameter is not supported. |
| 7F | 14 | 0x72 | **generalProgrammingFailure**  This NRC shall be returned if the server detects an error when writing to a memory location. |

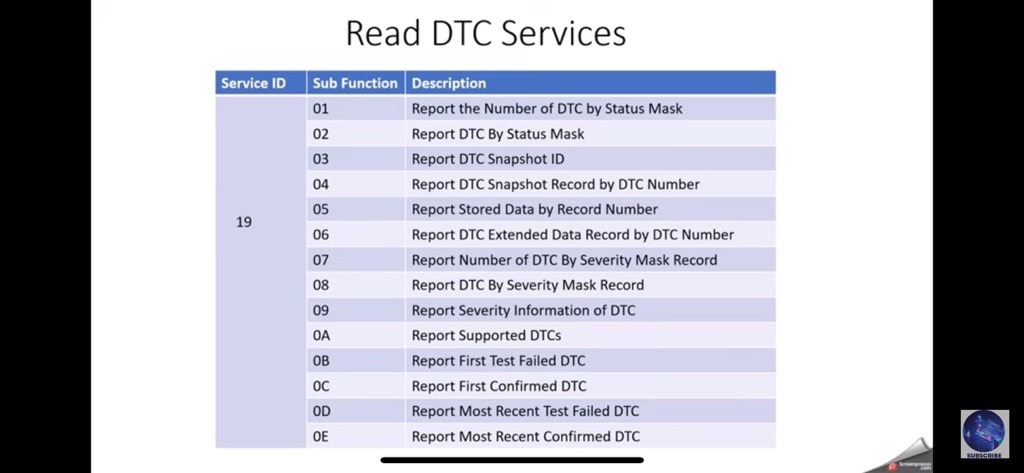
#### **ReadDTCInformation (19 hex)**

* The read DTC information service in UDS protocol to read the DTC’s from a vehicle or from a particular ECU or node. One of the major of the UDS protocol is fault diagnostics. Whenever any fault occurs in the vehicle, a diagnostic trouble code (DTC) corresponding to the fault is stored in the ECU fault code memory (FCM). The diagnostic service engineer can retrieve these DTCs by using the Read DTC Information (0x19) service.
* The Fault Diagnostics service allows the client to read both the emissions related or non-emission related DTC information. The client can define a status mask based on which the DTC information will be displayed.
* Except for DTC, to do more diagnostic which will help the diagnostic engineer to solve the problem, snapshot data also stored in the NVM. The DTC Snapshot data gives additional information about the engine’s parameters during the occurrence of the fault.

**Let’s see DTC status mask details:**

****

**This service has sub-functions that have been explained in the below table.**

****

**Note: many format is there here explained 1 example of request and response format , remains refer to UDS protocol doc.**

**Request Frame Format:**

|  |  |  |
| --- | --- | --- |
| **Data byte** | **Parameter Name** | **Byte Value** |
| #1 | ReadDTCInformation Request SID | 0x19 |
| #2 | sub-function = reportDTCByStatusMask | 0x02 |
| #3 | DTC status mask | 0x09 |

**Response Frame Format:**

There are two types of response frames. That is a **Positive response** and a **Negative response**.

* **Positive response**

|  |  |  |
| --- | --- | --- |
| **Data byte** | **Parameter Name** | **Byte Value** |
| #1 | ReadDTCInformation Response SID | 0x19 + 40 = 59 |
| #2 | sub-function = reportDTCByStatusMask | 0x02 |
| #3 | DTCStatusAvailabilityMask | 0x09 |
| #4  #n | DTCAndStatusRecord[ | 0x00 – 0xFF  :  0x00 – 0xFF |

* **Negative response**

The following negative response codes shall be implemented for this service

|  |  |  |  |
| --- | --- | --- | --- |
| **Response ID** | **SID** | **NRC** | **Description** |
| 7F | 19 | 0x12 | **sub-functionNotSupported**  This NRC shall be sent if the sub-function parameter is not supported. |
| 7F | 19 | 0x13 | **incorrectMessageLengthOrInvalidFormat**  This NRC shall be sent if the length of the message is wrong. |
| 7F | 19 | 0x31 | **requestOutOfRange ROOR**  This NRC shall be sent if:  ⎯ The client specified a DTCMaskRecord that was not recognized by the server;  ⎯ The client specified an invalid DTCSnapshotRecordNumber /  DTCExtDataRecordNumber. Note that this is to be differentiated from the case where  the DTCSnapshotRecordNumber and DTCMaskRecord combination or the  DTCExtDataRecordNumber and DTCMaskRecord combination is supported by the  server, but no data is currently associated with it (i.e., positive response required with  no data);  ⎯ The client specified a FunctionalGroupIdentifier that was not recognized by the server;  ⎯ The MemorySelection identifier was not recognized by the server. |

### **4. Input/Output Control**

#### **Input Output Control in UDS Protocol**

Input Output control service in UDS protocol is used in UDS to obtain an effective output. Input Output Control service ID is 0x2F and the Response is 0x6F.

This service is used by the client to substitute a value for an input signal, and internal server function, and force control to a value for an output of an electronic system. Generally, this service is used for input substitution and output control whereas routine Control is used if more complex input substitution/output control is necessary.

The request message contains the data Identifier to reference the input signal, internal server function, and force control of the output of the server. The vehicle manufacturer may require that the request message contain a control Enable Mask if the data identifier to be controlled references more than one parameter. If the vehicle manufacturer chooses to support the Enable Mask concept, then the control Enable Mask parameter is mandatory on all types of Input Output Control By Identifier requests for this service.

If input Output Control By Identifier is requested on a data Identifier that references a measured output value or feedback value, In order for the standard server control strategy to attempt to reach the desired state from the client request message, the server must substitute the proper target value within the server control strategy.

There is an optional byte. So, the tester can request to the server through optional bytes.

**There are 4 types of control parameters are used in the optional byte.**

1. **Return control to ECU**
2. **Reset to Default**
3. **Freeze current state**
4. **Short term adjustment**

**Return control to ECU**

This Return control to the ECU parameter tells the server that the client has no longer access and it gets back control of the mentioned signal.

**Reset to Default**

This Reset to Default parameter tells the server to set the default value to the mentioned input and output signal and internal parameters.

**Freeze current state**

This Freeze current state parameter tells the server to freeze mentioned input, output signal, and internal parameters to their current value.

**Short term adjustment**

This Short-term adjustment parameter tells the server to set its input, output signal, and internal parameters to the provided value.

**Request Frame Format:**

|  |  |  |
| --- | --- | --- |
| **Data byte** | **Parameter Name** | **Byte Value** |
| #1 | InputOutputControlByIdentifier Request SID | 0x2F |
| #2  #3 | dataIdentifier [ byte#1 ] = 0x9B  dataIdentifier [ byte#2 ] = 0x00 (“Air Inlet Door Position”) | 0x9B  0x00 |
| #4 | controlOptionRecord [ inputOutputControlParameter ] =  returnControlToECU | 0x00 |

**Response Frame Format:**

There are two types of response frames. That is a **Positive response** and a **Negative response**.

* **Positive response**

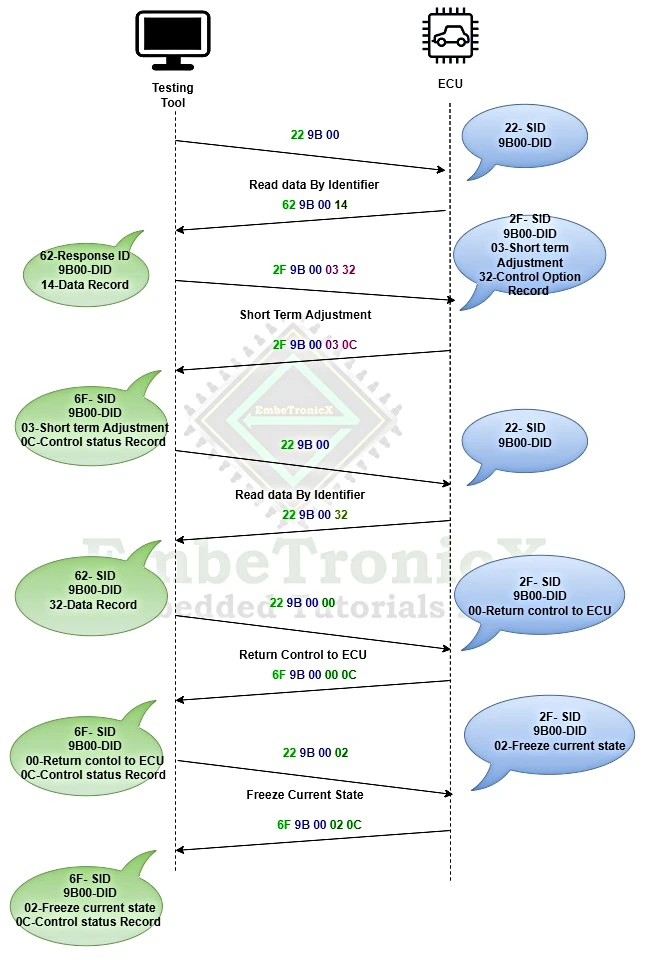
|  |  |  |
| --- | --- | --- |
| **Data byte** | **Parameter Name** | **Byte Value** |
| #1 | InputOutputControlByIdentifier Response SID | 0x2F + 40 = 0x6F |
| #2  #3 | dataIdentifier [ byte#1 ] = 0x9B  dataIdentifier [ byte#2 ] = 0x00 (“Air Inlet Door Position”) | 0x9B  0x00 |
| #4 | controlOptionRecord [ inputOutputControlParameter ] =  returnControlToECU | 0x00 |
| #5 | controlStatusRecord [ controlState#1 ] = 58% | 0x3A |

* **Negative response**

The following negative response codes shall be implemented for this service

|  |  |  |  |
| --- | --- | --- | --- |
| **Response ID** | **SID** | **NRC** | **Description** |
| 7F | 2F | 0x13 | **incorrectMessageLengthOrInvalidFormat**  This NRC shall be sent if the length of the message is wrong. |
| 7F | 2F | 0x22 | **conditionsNotCorrect**  This NRC shall be returned if the criteria for the request InputOutputControl are not met. |
| 7F | 2F | 0x31 | **requestOutOfRange ROOR**  **This NRC shall be sent if:**  ⎯ the requested dataIdentifier value is not supported by the device;  ⎯ the value contained in the inputOuptputControlParameter is invalid (see definition of inputOutputControlParameter);  ⎯ one or multiple of the applicable controlState values of the controlOptionRecord record are invalid;  ⎯ the combination of bits enabling control in the ControlEnableMaskRecord is not supported by the device; |
| 7F | 2F | 0x33 | **securityAccessDenied**  This NRC shall be returned if a client sends a request with a valid secure dataIdentifier and the  server’s security feature is currently active. |

**Example:**

****

### **Remote activation of routine**

This functional unit specifies the services of remote activation of routines, as they shall be implemented in servers and clients. It has the below service,

#### **Routine control service(0x31)**

The routine control service ID is 0x31 and the response SID is 0x71.

The Routine Control (0x31) Service is one of the important diagnostic services defined by the ISO 14229-1 standard. Routines can be used to allow the testing tool to do some specific functions in the ECU. With the help of the Routine Control service, the client(tester) can start, stop and check the result of the routine produced after the successful execution of the routine.

The Routine Control service is used by the client during vehicle testing some system testing may run over a certain period of time.

To perform the testing a routine may be triggered by the client externally or a routine needs to be started by the client in the server’s memory. Before starting the routine some of the pre-conditions like the engine should be turned off, the vehicle should not move, immobilized should be unlocked, etc., needs to be checked and the ECU needs to switch to the appropriate diagnostic session and the security needs to be unlocked. Once after the start of the routine, the ECU will send a positive response to the client.

Suppose if the tester wants to stop the routine there are two methods to stop the routine,

**The client interrupts the routine to stop.**

**The server or ECU finishes the routine after a specified time frame.**

Using the Routine Control service, the client (tester) can start, stop and check the result of the routine produced after the successful execution of the routine. If the tester wants to know the result of the routine with the help of the request routine result sub-function, we will get the appropriate result.

A routine is a series of actions that the ECU uses to carry out a certain task, such as validating the programming parameters, wiping the flash memory, or ensuring the accuracy of the data. Every routine has a special identifier (RID) that describes its purpose and inputs. Testers should always refer to OEM documentation for details on how to safely and effectively use the Routine Control (0x31) service.

**There are a few examples of tasks that are using Routine Control Services,**

* Programming or re-Programming of the ECU.
* Erasing the fault Memory.
* Reading or writing of calibration or configuration data from/to the ECU.
* Performing Self-test and functional tests.
* Activating or De-Activating the particular feature in the ECU.

**Routine control service has the below-supported sub-function,**

|  |  |
| --- | --- |
| **sub-functions** | **Description** |
| Start Routine(0x01) | The start Routine initiates the execution of a routine corresponding to the given routine identifier. Before using the Start routine it may be necessary to switch the server to a specific diagnostic session using the Diagnostic Session Control service or to unlock the server with secure access. Once the server receives a request to start the routine from the client, the routine shall be started in the server’s memory. If the routine is started successfully it will send a positive response otherwise it will send a negative response to the client. |
| Stop Routine(0x02) | The server routine shall be stopped in the server’s memory after the completion of the Stop routine request message. Based on the application software the server routine shall be stopped at any time or with initialized time in the server’s memory. |
| Request Routine Result(0x03) | This Sub-function gives the exit status information of the completed routine.  For example, if the routine is exited normally, it will send a positive response with a normal exit with the result. If the routine is exited abnormally it will return a negative response with an abnormal exit with result. |

**Request Frame Format:**

|  |  |  |
| --- | --- | --- |
| **Data byte** | **Parameter Name** | **Byte Value** |
| #1 | RoutineControl Request SID | 0x31 |
| #2 | sub-function | 0x01 – 0x03 |
| #3  #4 | routineIdentifier [] = [  byte#1 (MSB)  byte#2 (LSB) ] | 0x00 – 0xFF  0x00 – 0xFF |
| #5  :  #n | routineControlOptionRecord[] = [  routineControlOption#1  :  routineControlOption#m ] | 0x00 – 0xFF  :  0x00 – 0xFF |

**Response Frame Format:**

There are two types of response frames. That is a **Positive response** and a **Negative response**.

* **Positive response**

|  |  |  |
| --- | --- | --- |
| **Data byte** | **Parameter Name** | **Byte Value** |
| #1 | RoutineControl Response SID | 0x31 + 40 = 71 |
| #2 | sub-function | 0x01 – 0x03 |
| #3  #4 | routineIdentifier [] = [  byte#1 (MSB)  byte#2 (LSB) ] | 0x00 – 0xFF  0x00 – 0xFF |
| #5 | routineInfo | 0x00 – 0xFF |
| #6  :  #n | routineStatusRecord[] = [  routineStatus#1  :  routineStatus#m ] | 0x00 – 0xFF  :  0x00 – 0xFF |

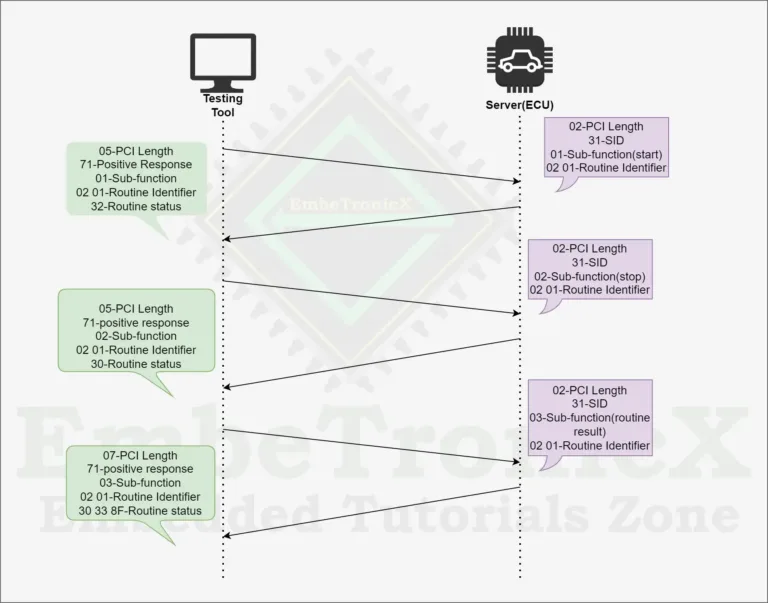
* **Negative response**

The following negative response codes shall be implemented for this service

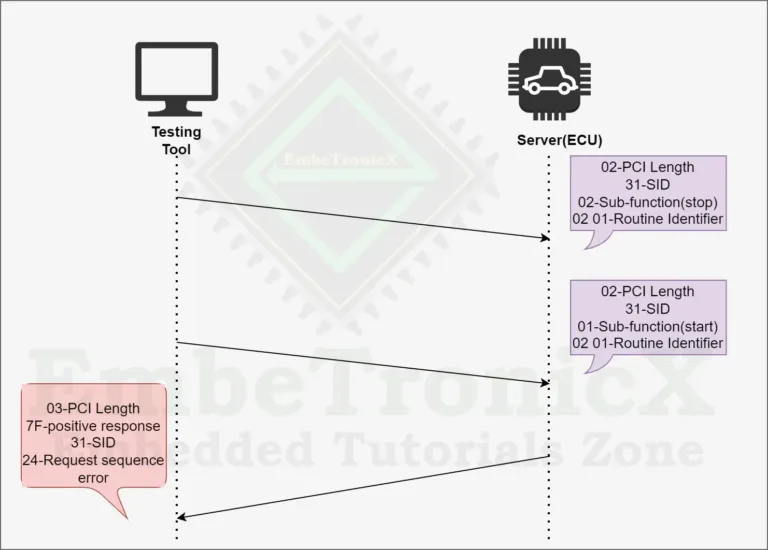
|  |  |  |  |
| --- | --- | --- | --- |
| **Response ID** | **SID** | **NRC** | **Description** |
| 7F | 31 | 0x12 | **sub-functionNotSupported**  This NRC shall be sent if the requested sub-function is either generally not supported or is  not supported for the requested RoutineIdentifier. |
| 7F | 31 | 0x13 | **incorrectMessageLengthOrInvalidFormat**  This NRC shall be sent if the length of the message is wrong. |
| 7F | 31 | 0x22 | **conditionsNotCorrect**  This NRC shall be returned if the criteria for the request InputOutputControl are not met. |
| 7F | 31 | 0x24 | **requestSequenceError**  This NRC shall be returned if  the routine is currently active and can not be restarted when the 'startRoutine' sub-function is received (it is up to the vehicle manufacturer whether a given routine can be restarted while active),  the routine is not currently active when the 'stopRoutine' sub-function is received,  routine results are not available when the 'requestRoutineResults' sub-function is received (e.g., the requested routineIdentifier has never been started). |
| 7F | 31 | 0x31 | **requestOutOfRange**  This NRC shall be returned if:  ⎯The server does not support the requested routineIdentifier,  ⎯The user optional routineControlOptionRecord contains invalid data for the requested routineIdentifier. |
| 7F | 31 | 0x33 | **securityAccessDenied**  This NRC shall be returned if a client sends a request with a valid secure dataIdentifier and the  server’s security feature is currently active. |
| 7F | 31 | 0x72 | **GeneralProgrammingFailure**  This NRC shall be returned if the server detects an error when performing a routine, which accesses server internal memory. An example is when the routine erases or programs a  certain memory location in the permanent memory device (e.g. Flash Memory) and the access to that memory location fails. |

**Example:**

**For Positive responses,**

****

**For Negative responses,**

****

### **Upload/Download**

This Upload Download Functional Unit in UDS Protocol specifies the services of how the data upload and downloads take place between the Client (testing tool) and server(ECU) and vice versa. Mainly these services are used for flashing the new firmware in ECU and downloading data from the ECU. This functional unit supports the below service,

1. **Request Download (0x34) service**
2. **Request Upload (0x35) service**
3. **Transfer data (0x36) service**
4. **Request transfer exit (0x37) service**
5. **RequestFileTransfer (0x38) service**

**Let’s explain in detail about each service:**

#### **Request Download (0x34) service**

The Request Download Service ID is 0x34 and the Response ID is 0x74.

Request Download service is one of the most important services among all services in the UDS protocol. This service is used to send the data to the ECU from the testing tool. This service needs the below parameter to download the data and it needs to be given from the client side. Those parameters are,

* Memory location where the data needs to be downloaded in the ECU
* Size of the data

The server shall perform all required actions to retrieve data before sending a positive message once it receives the request Download Request Message.

**Request Frame Format:**

|  |  |  |
| --- | --- | --- |
| **Data byte** | **Parameter Name** | **Byte Value** |
| #1 | RequestDownload Request SID | 0x34 |
| #2 | dataFormatIdentifier | 0x00 – 0xFF |
| #3 | addressAndLengthFormatIdentifie | 0x00 – 0xFF |
| #4  :  #(m-1)+4 | memoryAddress[] = [  byte#1 (MSB)  :  byte#m ] | 0x00 – 0xFF  :  0x00 – 0xFF |
| #n-(k-1)  :  #n | memorySize[] = [  byte#1 (MSB)  :  byte#k ] | 0x00 – 0xFF  :  0x00 – 0xFF |

**Response Frame Format:**

There are two types of response frames. That is a **Positive response** and a **Negative response**.

* **Positive response**

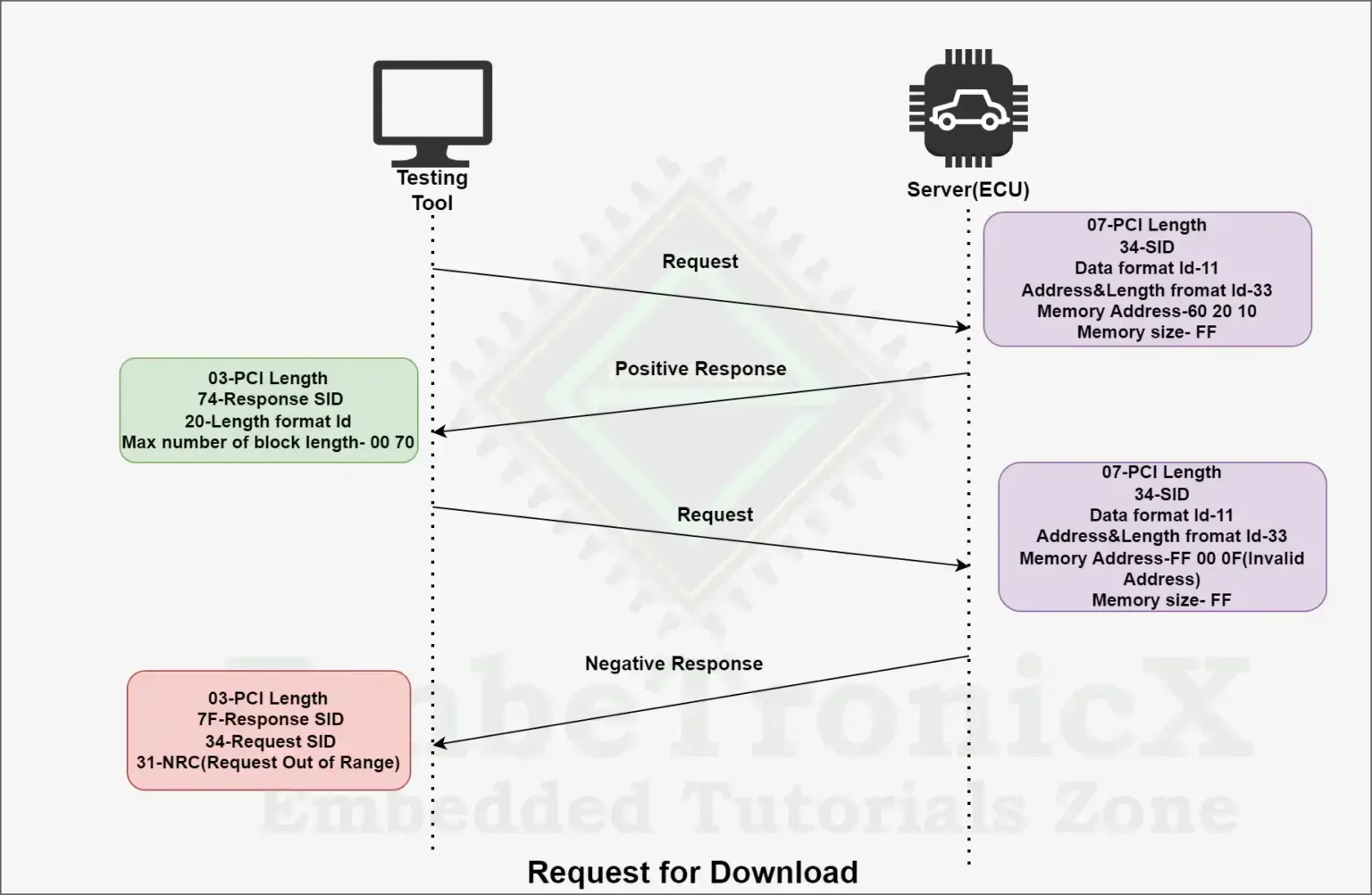
|  |  |  |
| --- | --- | --- |
| **Data byte** | **Parameter Name** | **Byte Value** |
| #1 | RequestDownload Response SID | 0x34 + 40 = 74 |
| #2 | lengthFormatIdentifier | 0x00 – 0xFF |
| #3  :  #n | maxNumberOfBlockLength = [  byte#1 (MSB)  :  byte#m ] | 0x00 – 0xFF  :  0x00 – 0xFF |

* **Negative response**

The following negative response codes shall be implemented for this service

|  |  |  |  |
| --- | --- | --- | --- |
| **Response ID** | **SID** | **NRC** | **Description** |
| 7F | 34 | 0x13 | **incorrectMessageLengthOrInvalidFormat**  This NRC shall be sent if the length of the message is wrong. |
| 7F | 34 | 0x22 | **conditionsNotCorrect**  This NRC shall be returned if a server receives a request for this service while in the process of receiving a download of a software or calibration module. This could occur if there is a data size mismatch between the server and the client during the download of a module. |
| 7F | 34 | 0x31 | **requestOutOfRange**  This NRC shall be returned if:  ⎯the specified dataFormatIdentifier is not valid.  ⎯the specified addressAndLengthFormatIdentifier is not valid.  ⎯the specified memoryAddress/memorySize is not valid. |
| 7F | 34 | 0x33 | **securityAccessDenied**  This NRC shall be returned if the server is secure (for server’s that support the  SecurityAccess service) when a request for this service has been received. |
| 7F | 34 | 0x70 | **uploadDownloadNotAccepted**  This NRC indicates that an attempt to download to a server's memory cannot be accomplished due to some fault conditions. |

**Example:**



#### **Request Upload (0x35) service**

The Transfer data Service ID is 0x35 and the Response ID is 0x75

The Request Upload service is used by the client to initiate the transfer of data from the ECU to the client. Once the server receives the request upload service request message, the server takes any necessary action to send the data before sending a positive response message.

**Request Frame Format:**

|  |  |  |
| --- | --- | --- |
| **Data byte** | **Parameter Name** | **Byte Value** |
| #1 | RequestUpload Request SID | 0x35 |
| #2 | dataFormatIdentifier | 0x00 – 0xFF |
| #3 | addressAndLengthFormatIdentifie | 0x00 – 0xFF |
| #4  :  #(m-1)+4 | memoryAddress[] = [  byte#1 (MSB)  :  byte#m ] | 0x00 – 0xFF  :  0x00 – 0xFF |
| #n-(k-1)  :  #n | memorySize[] = [  byte#1 (MSB)  :  byte#k ] | 0x00 – 0xFF  :  0x00 – 0xFF |

**Response Frame Format:**

There are two types of response frames. That is a **Positive response** and a **Negative response**.

* **Positive response**

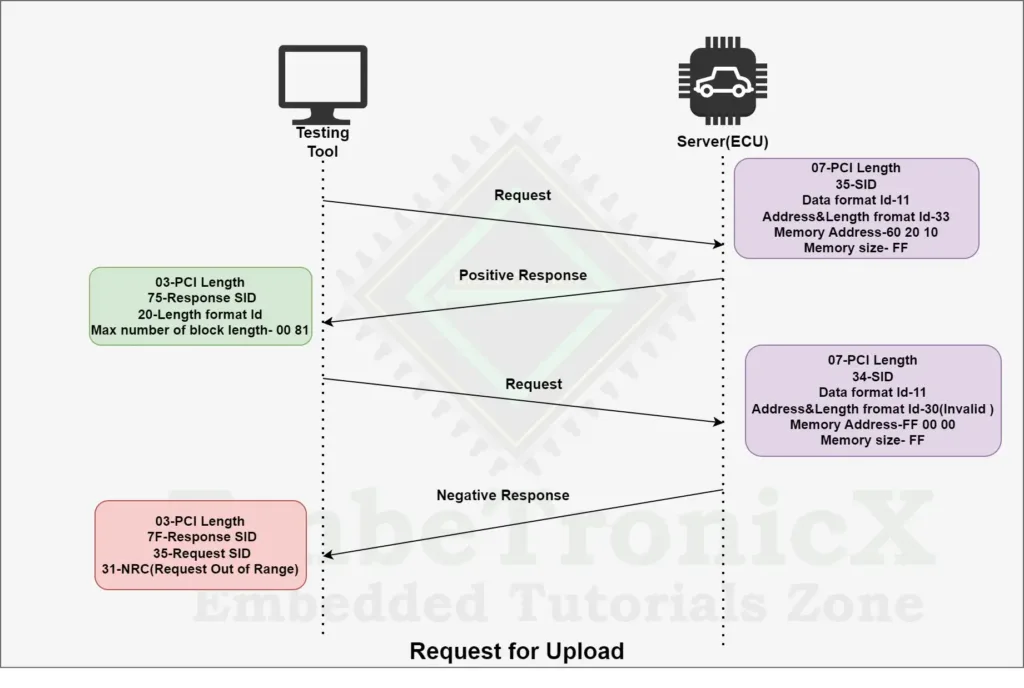
|  |  |  |
| --- | --- | --- |
| **Data byte** | **Parameter Name** | **Byte Value** |
| #1 | RequestUpload Response SID | 0x35 + 40 = 75 |
| #2 | lengthFormatIdentifier | 0x00 – 0xFF |
| #3  :  #n | maxNumberOfBlockLength = [  byte#1 (MSB)  :  byte#m ] | 0x00 – 0xFF  :  0x00 – 0xFF |

* **Negative response**

The following negative response codes shall be implemented for this service

|  |  |  |  |
| --- | --- | --- | --- |
| **Response ID** | **SID** | **NRC** | **Description** |
| 7F | 35 | 0x13 | **incorrectMessageLengthOrInvalidFormat**  This NRC shall be sent if the length of the message is wrong. |
| 7F | 35 | 0x22 | **conditionsNotCorrect**  This NRC shall be returned if a server receives a request for this service while in the process of receiving a download of a software or calibration module. This could occur if there is a data size mismatch between the server and the client during the download of a module. |
| 7F | 35 | 0x31 | **requestOutOfRange**  This NRC shall be returned if:  ⎯the specified dataFormatIdentifier is not valid.  ⎯the specified addressAndLengthFormatIdentifier is not valid.  ⎯the specified memoryAddress/memorySize is not valid. |
| 7F | 35 | 0x33 | **securityAccessDenied**  This NRC shall be returned if the server is secure (for server’s that support the SecurityAccess service) when a request for this service has been received. |
| 7F | 35 | 0x70 | **uploadDownloadNotAccepted**  This NRC indicates that an attempt to download to a server's memory cannot be accomplished due to some fault conditions. |

**Example:**



#### **Transfer data (0x36) service**

The Transfer data Service ID is 0x36 and the Response ID is 0x76.

Transfer data Service is used to transfer the data from the client to ECU and ECU to the client, which means this service is used for both uploading and downloading of data. This service is used together with either request upload or request download.

**For example,**

* If the client initiated a request upload service, the data to be uploaded is included in the parameters transfer response parameter in the Transfer Data response messages.
* If the client initiated a request download service, the data to be downloaded is included in the parameters transfer response parameter in the Transfer Data response messages.

To improve error handling in the transfer data service, the block sequence counter parameter is the request message, initial value of the block sequence counter will be set to 1 when receiving a request upload and request download request message.

**Request Frame Format:**

|  |  |  |
| --- | --- | --- |
| **Data byte** | **Parameter Name** | **Byte Value** |
| #1 | TransferData Request SID | 0x36 |
| #2 | blockSequenceCounter | 0x00 – 0xFF |
| #3  :  #n | transferRequestParameterRecord[] = [  transferRequestParameter#1  :  transferRequestParameter#m ] | 0x00 – 0xFF  :  0x00 – 0xFF |

**Response Frame Format:**

There are two types of response frames. That is a **Positive response** and a **Negative response**.

* **Positive response**

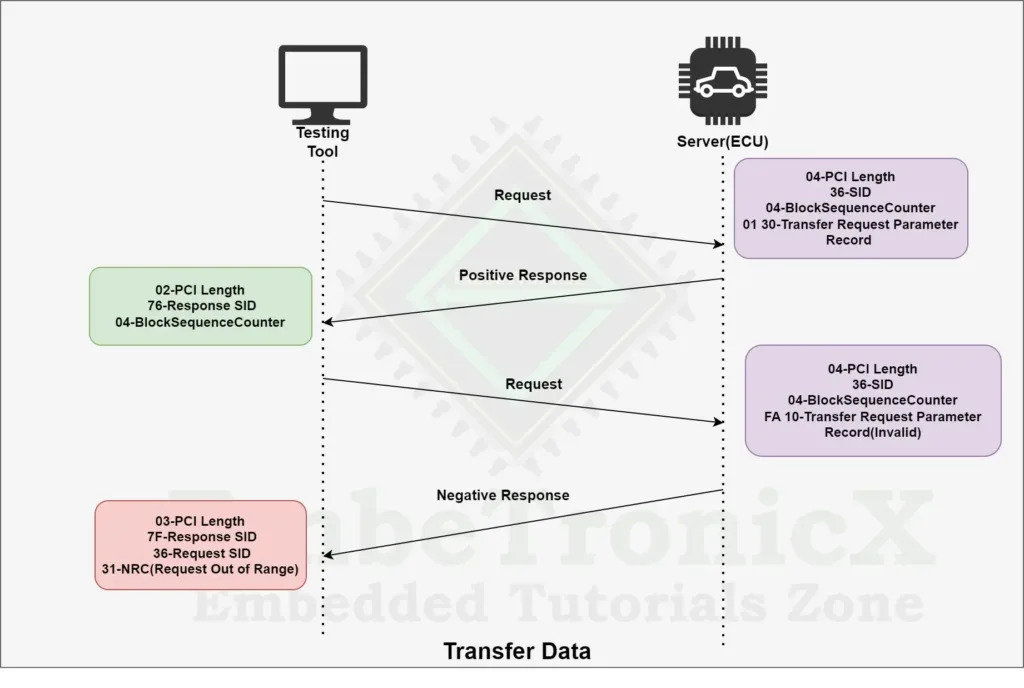
|  |  |  |
| --- | --- | --- |
| **Data byte** | **Parameter Name** | **Byte Value** |
| #1 | TransferData Response SID | 0x36 + 40 = 76 |
| #2 | blockSequenceCounter | 0x00 – 0xFF |
| #3  :  #n | transferResponseParameterRecord[] = [  transferResponseParameter#1  :  transferResponseParameter#m ] | 0x00 – 0xFF  :  0x00 – 0xFF |

* **Negative response**

The following negative response codes shall be implemented for this service

|  |  |  |  |
| --- | --- | --- | --- |
| **Response ID** | **SID** | **NRC** | **Description** |
| 7F | 36 | 0x13 | **incorrectMessageLengthOrInvalidFormat**  This NRC shall be sent if the length of the message is wrong.(e.g., message length does not meet requirements of maxNumberOfBlockLength parameter returned in the positive  response to the requestDownload service). |
| 7F | 36 | 0x24 | **requestSequenceError**  The server shall use this response code:  ⎯If the RequestDownload or RequestUpload service is not active when a request for this service is received;  ⎯If the RequestDownload or RequestUpload service is active, but the server has already received all data as determined by the memorySize parameter in the active  RequestDownlod or RequestUpload service;  NOTE The repetition of a TransferData request message with a  blockSequenceCounter equal to the one included in the previous TransferData request message shall be accepted by the server. |
| 7F | 36 | 0x31 | **requestOutOfRange**  This NRC shall be returned if:  ⎯The transferRequestParameterRecord contains additional control parameters (e.g. additional address information) and this control information is invalid.  ⎯The transferRequestParameterRecord is not consistent with the requestDownload or requestUpload service parameter maxNumberOfBlockLength.  ⎯The transferRequestParameterRecord is not consistent with the server’s memory alignment constraints. |
| 7F | 36 | 0x71 | **securityAccessDenied**  This NRC shall be returned if the server is secure (for server’s that support the SecurityAccess service) when a request for this service has been received. |
| 7F | 36 | 0x72 | **generalProgrammingFailure**  This NRC shall be returned if the server detects an error when erasing or programming a memory location in the permanent memory device (e.g. Flash Memory) during the download of data. |
| 7F | 36 | 0x73 | **wrongBlockSequenceCounter**  This NRC shall be returned if the server detects an error in the sequence of the blockSequenceCounter. NOTE The repetition of a TransferData request message with a  blockSequenceCounter equal to the one included in the previous TransferData request message shall be accepted by the server. |
| 7F | 36 | 0x92 / 0x93 | **voltageTooHigh / voltageTooLow**  This return code shall be sent as applicable if the voltage measured at the primary power pin of the server is out of the acceptable range for downloading data into the server’s  permanent memory (e.g. Flash Memory). |

**Example:**



#### **Request transfer exit (0x37) service**

The Request Transfer Exit Service ID is 0x37 and the Response ID is 0x77.

This service is used to terminate the data transfer between the client and the server both request upload and request download.

**Request Frame Format:**

|  |  |  |
| --- | --- | --- |
| **Data byte** | **Parameter Name** | **Byte Value** |
| #1 | RequestTransferExit Request SID | 0x37 |
| #2  :  #n | transferRequestParameterRecord[] = [  transferRequestParameter#1  :  transferRequestParameter#m ] | 0x00 – 0xFF  :  0x00 – 0xFF |

**Response Frame Format:**

There are two types of response frames. That is a **Positive response** and a **Negative response**.

* **Positive response**

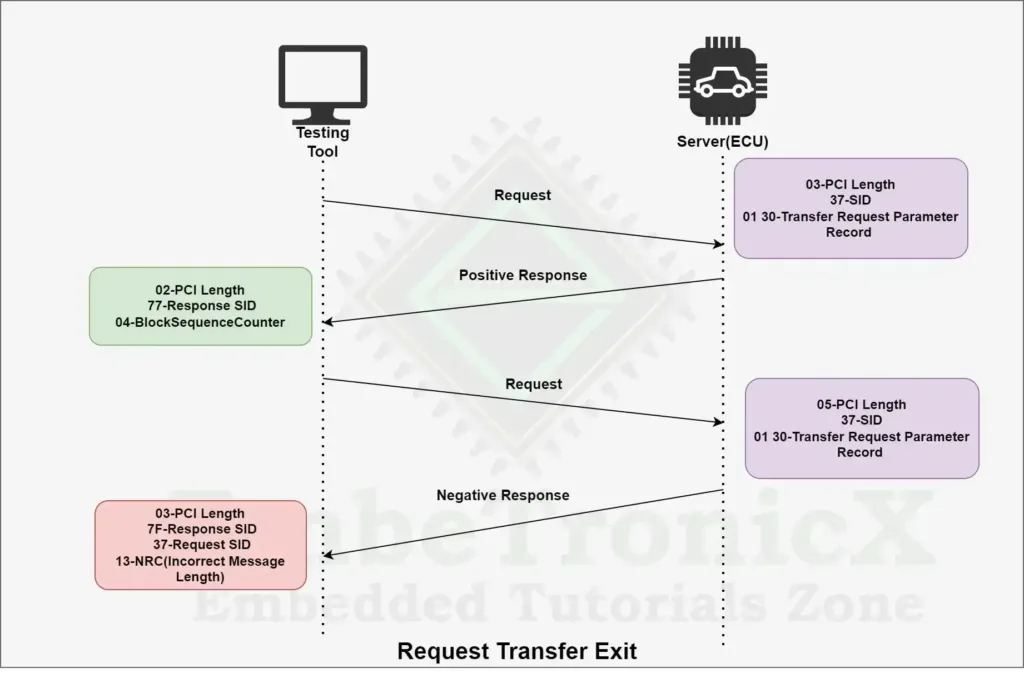
|  |  |  |
| --- | --- | --- |
| **Data byte** | **Parameter Name** | **Byte Value** |
| #1 | RequestTransferExit Response SID | 0x37 + 40 = 77 |
| #2  :  #n | transferResponseParameterRecord[] = [  transferResponseParameter#1  :  transferResponseParameter#m ] | 0x00 – 0xFF  :  0x00 – 0xFF |

* **Negative response**

The following negative response codes shall be implemented for this service

|  |  |  |  |
| --- | --- | --- | --- |
| **Response ID** | **SID** | **NRC** | **Description** |
| 7F | 37 | 0x13 | **incorrectMessageLengthOrInvalidFormat**  This NRC shall be sent if the length of the message is wrong.(e.g., message length does not meet requirements of maxNumberOfBlockLength parameter returned in the positive  response to the requestDownload service). |
| 7F | 37 | 0x24 | **requestSequenceError**  This NRC shall be returned if:  ⎯The programming process is not completed when a request for this service is received;  ⎯The RequestDownload or RequestUpload service is not active; |
| 7F | 37 | 0x31 | **requestOutOfRange**  This NRC shall be returned if the transferRequestParameterRecord contains invalid data. |
| 7F | 37 | 0x72 | **generalProgrammingFailure**  This NRC shall be returned if the server detects an error when finalizing the data transfer between the client and server (e.g., via an integrity check). |

**Example:**



#### **RequestFileTransfer (0x38) service**

The Request file Transfer Service ID is 0x38 and the Response ID is 0x78.

The Request File Transfer service is used by the client to initiate a file data transfer from the client to the server or from the server to the client. The ability to get file system data is another feature of this service.

If a server implements a file system for data storage, this request file transfer service is an alternate solution for the Request download and Request upload services. The Request File Transfer service should be used in place of request download or request upload when configuring a download or upload service to or from a file system. The actual data transfer and termination of the data transfer are implemented by using the Transfer data and Request Transfer Exit as used with the request download or request upload service.

**Request Frame Format:**

|  |  |  |
| --- | --- | --- |
| **Data byte** | **Parameter Name** | **Byte Value** |
| #1 | RequestFileTransfer Request SID | 0x38 |
| #2 | modeOfOperation | 0x01 – 0x05 |
| #3  #4 | filePathAndNameLength [  byte#1 (MSB)  byte#2] (LSB) | 0x00 – 0xFF  :  0x00 – 0xFF |
| #5  :  #5+n-1 | filePathAndName = [  byte#1 (MSB)  :  byte#n ] | 0x00 – 0xFF  :  0x00 – 0xFF |
| #5+n | dataFormatIdentifier | 0x00 – 0xFF |
| #5+n+1 | fileSizeParameterLength | 0x00 – 0xFF |
| #5+n+2  :  #5+n+2+k-1 | fileSizeUnCompressed= [  byte#1 (MSB)  :  byte#k ] | 0x00 – 0xFF  :  0x00 – 0xFF |
| #5+n+2+k  :  #5+n+1+2k | fileSizeCompressed= [  byte#1 (MSB)  :  byte#k ] | 0x00 – 0xFF  :  0x00 – 0xFF |

**Response Frame Format:**

There are two types of response frames. That is a **Positive response** and a **Negative response**.

* **Positive response**

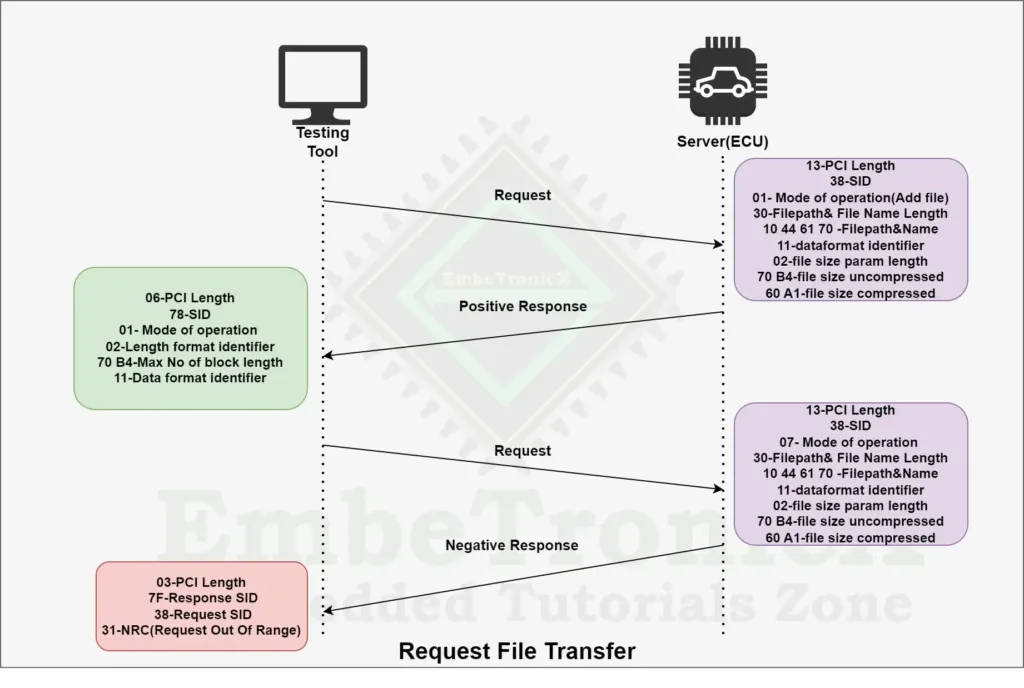
|  |  |  |
| --- | --- | --- |
| **Data byte** | **Parameter Name** | **Byte Value** |
| #1 | RequestFileTransfer Response SID | 0x38 + 40 = 78 |
| #2 | modeOfOperation | 0x01 – 0x05 |
| #3 | lengthFormatIdentifier | 0x00 – 0xFF |
| #4  :  #4+(m-1) | maxNumberOfBlockLength = [  byte#1 (MSB)  :  byte#m] | 0x00 – 0xFF  :  0x00 – 0xFF |
| #4+m | dataFormatIdentifier | 0x00 – 0xFF |
| #4+m+1  #4+m+2 | fileSizeOrDirInfoParameterLength [  byte#1 (MSB)  byte#2 (LSB)] | 0x00 – 0xFF  :  0x00 – 0xFF |
| #4+m+3  :  #4+m+3+k-1 | fileSizeUncompressedOrDirInfoLength= [  byte#1 (MSB)  :  byte#k] | 0x00 – 0xFF  :  0x00 – 0xFF |
| #4+m+3+k  :  #4+m+3+2k-1 | fileSizeCompressed= [  byte#1 (MSB)  :  byte#k ] | 0x00 – 0xFF  :  0x00 – 0xFF |

* **Negative response**

The following negative response codes shall be implemented for this service

|  |  |  |  |
| --- | --- | --- | --- |
| **Response ID** | **SID** | **NRC** | **Description** |
| 7F | 38 | 0x13 | **incorrectMessageLengthOrInvalidFormat**  This NRC shall be sent if the length of the message is wrong. |
| 7F | 38 | 0x22 | **conditionsNotCorrect**  This NRC shall be returned if a server receives a request for this service while in the process  of downloading or uploading data or other conditions to be able to execute this service are not met. |
| 7F | 38 | 0x31 | **requestOutOfRange**  This NRC shall be returned if:  ⎯The specified dataFormatIdentifier is not valid  ⎯The specified modeOfOperation is not valid  ⎯The specified fileSizeParameterLength is not valid  ⎯The specified filePathAndNameLength is not valid  ⎯The specified fileSizeUncompressed is not valid  ⎯The specified fileSizeCompressed is not valid  ⎯The specified filePathAndName is not valid |
| 7F | 38 | 0x33 | **securityAccessDenied**  This NRC shall be returned if the server is secure (for server’s that support the SecurityAccess service) when a request for this service has been received. |
| 7F | 38 | 0x70 | **uploadDownloadNotAccepted**  This NRC indicates that an attempt to download to a server's memory cannot be accomplished due to some fault conditions. |

**Example:**

****

## **Negative Response Codes (NRC): UDS Protocol**

UDS NRC (Negative Response Code) stands for Unified Diagnostic Services Negative Response Code. It is a critical component of the UDS (Unified Diagnostic Services) protocol, which is widely used in the automotive industry for diagnostic communication between electronic control units (ECUs) in vehicles. UDS NRC codes are used to convey information about the success or failure of diagnostic services and requests within the UDS protocol.

all the Negative Response Codes (NRC) used in the IS0-14229 standard for identifying the internal protocol fault for not following the standard are defined below. Each diagnostic service identifier has defined its negative response codes for that particular fault. The diagnostic service implementation within the server can also utilize additional and applicable Negative Response Codes (NRC) laid out in this as defined by the vehicle manufacturer also.

**All the negative response codes are split into three categorized from0x00 – 0xFFranges, such as:**

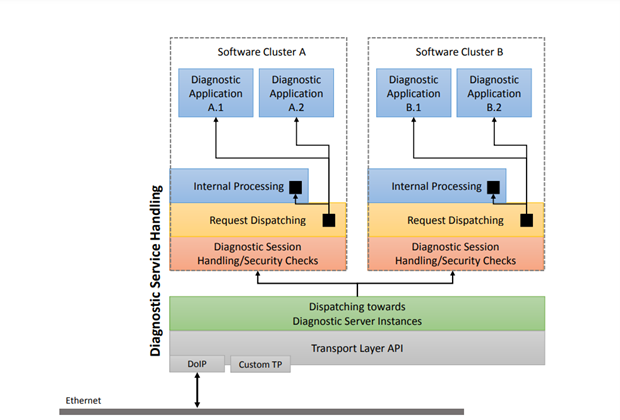
* **0x00:** The purpose of this Negative Response Codes (NRC) is to available for a positive response parameter value for server internal implementation.
* **0x01 – 0x7F:** This range is used for the communication-related negative response codes.
* **0x80 – 0xFF:** This range is used as the Negative Response Codes (NRC) for specific conditions that are not correct at that instant of time when the request is received by the server. These response codes could also be utilized whenever the response code 0x22 is listed as valid. By the way to report more specifically why the requested action cannot be taken for a positive response.

**Below listed all NRC Codes:**

|  |  |  |
| --- | --- | --- |
| **NRC Code Value** | **NRC Code Name** | **Description** |
| **0x00** | Positive Response | The 0x00 is a special and initiative value that cannot be used for ant NRC. It is this parameter value reserved for several internal code implementation reasons. |
| **0x01 – 0x0F** | ISO SAE Reserved | This range of NRC is reserved for the future definition that maybe increase the diagnostic functionality more. |
| **0x10** | General Reject | If there is none of the NRC is available in UDS standard document to meet the needs of the implementation, then you can use this NRC for your ECU. |
| **0x11** | Service Not Supported | Suppose the client has sent a request with a Service Identifier. That is not supporting in your ECU by your OEM. Even if you can also say as per the requirement or as per the OEM. But the service engineer or client has requested that unknown SID request to the server. Then the server will send the 0x11 NRC response message to the client that it is an unknown Service Identifier. |
| **0x12** | Sub-function Not Supported | Suppose the client has sent a valid and known Service identifier. But the Sub-function Identifier is unknown or invalid. then the server will send 0x12 Negative Response Code to the client. |
| **0x13** | Incorrect Message Length Or Invalid Format | If the client has sent any diagnostic request message whose length or the frame format does not match with the prescribed length or the format for that specified service (SID), then the server will send this NRC. |
| **0x14** | Response Too Long | If the client has sent a diagnostic request and the response message length is more than the Transport protocol maximum length, then the server will send this NRC. The CAN-TP has 4096 bytes has a maximum length, if the server will send more than that, then this NRC will be sent by the server to the client. Ex: suppose you requested more number of DIDs at a time to read the data if it will cross that limit, then this NRC will get generated by the server. |
| **0x15 – 0x20** | ISO SAE Reserved | This range of NRC is also reserved for the future definition that maybe increase the diagnostic functionality more. |
| **0x21** | Busy Repeat Request | If the server is too busy with some other operation or you can say any other client request in a multi-client environment, then the server will send this NRC to the client. This NRC is supported by each SID.  Ex: Suppose there are two clients connected to your vehicle simultaneously. One client has already requested a service that is under process means incomplete and the second client is requesting another request to the server then the server will send the NRC 0x21 to the second server to wait for some time. This time will be defined in the document that you need to check. |
| **0x22** | Conditions Not Correct | Before proceeding any service from the client, the server will check for prerequisite conditions are met or not. If not then the server will send this NRC.  Ex: you can say the ECU operational low threshold or high threshold voltage, temperature, etc. |
| **0x23** | ISO SAE Reserved | This NRC is also reserved for the future definition. |
| **0x24** | Request Sequence Error | If the client will send the diagnostic message non-sequentially, then the server will send this NRC to the client.  Ex: Suppose in security access SID, the client sent security key first and then seed request, then the server will send this NRC. because in this 0x27 service, the client should send the seed request first and then the security key. |
| **0x25** | No Response from Sub-net Component | Suppose the diagnostic request service sent by the client to the server, but the received ECU is not the target ECU and it is a gateway. So that the Gateway ECU will send this request to the target ECU and that should perform the operation and send back the response message to the Gateway ECU, and then it will send the response message. If the Gateway ECU does not receive the response message from that target ECU or server then the Gateway ECU will send this NRC to the client.  This NRC supported by each SID. |
| **0x26** | Failure Prevents Execution Of Requested Action | If the client requested service and at that time a fault occurred in server ECU after receiving of this ECU so that at least one DTC status bit for TestFailed, Pending, Confirmed or TestFailedSinceLastClear set to 1, then the server will send this NRC to the client. Basically this failure condition prevents the server from performing the requested action so that the server will not able to execute the client request. |
| **0x27 – 0x30** | ISO SAE Reserved | This NRC is also reserved for the future definition. |
| **0x31** | Request Out Of Range | If the server received a client request with a parameter (DID, RID) that is out of range, then the server will send this NRC. Suppose in your ECU, the DID or RID having some range used and the client requesting it which is not supported in your ECU or might be not supported in this active session, it will send this NRC. |
| **0x32** | ISO SAE Reserved | This NRC is also reserved for the future definition. |
| **0x33** | Security Access Denied | The server will send this NRC if the security strategy is not satisfied with the server. The Server will send this NRC if any of the below conditions will not satisfy: Server test conditions are not met. Service message sequence (first diagnostic session and then security access service request). The server needs to unlock by the client for some read or write access to the server, but without unlock, a client trying to access the protected memory. |
| **0x34** | ISO SAE Reserved | This NRC is also reserved for the future definition. |
| **0x35** | Invalid Key | If the server received a security key from the client that is not matching with the server-generated key, then the server will send this NRC. |
| **0x36** | exceed Number Of Attempts | Basically, in each server, there will be a security retrieval counter and it will be having a limit like 3 or 5 times. So if a client will send the wrong security key more than the defined counter value, the server will send this NRC to the client. |
| **0x37** | Required Time Delay Not Expired | Once the client will send a wrong security key, client should wait the defined time, and then it should send the key again. But if the client will send the security key before that then the server will send this NRC. |
| **0x38 – 0x4F** | Reserved By Extended Data Link Security Document | This is reserved for future implementation of security layer related NRC |
| **0x50 – 0x6F** | ISO SAE Reserved | This NRC is also reserved for the future definition. |
| **0x70** | Upload Download Not Accepted | Sometimes due to some fault conditions in the server or server memory, it will not accept any upload or download request from the client. |
| **0x71** | Transfer Data Suspended | During the data transfer, sometimes due to some failure of any kind of fault, the server will not able to write data onto the memory. So in that situation, the server will send this NRC to the client. |
| **0x72** | General Programming Failure | If the server detects an error during the erasing or programming of the memory location of permanent memory (e.g. NVM/Flash/EEPROM), then the server will send this NRC to the client. |
| **0x73** | Wrong Block Sequence Counter | Basically whenever a multi-frame consecutive data packets sent by the client to server. In the Consecutive Frame, there will be a Frame index that will increase in each next frame. The server also receives that and compares it with the previous value that should be +1 in each current frame received. If there is any miss match, the server will send this NRC to the client. |
| **0x74 – 0x77** | ISO SAE Reserved | This NRC is also reserved for the future definition. |
| **0x78** | Request Correctly Received-Response Pending | This NRC indicates that the request message from client successfully received by the server. All the requested parameters are also valid. But the server is not ready or due to busy or it is taking some more time to execute the client request in the defined server parameter. So if the client sends another request or the server P2 Client time out happened, the server will send this NRC to the client to inform that to wait for some more time period nothing but the P2\* Client timing value. After the execution, the server will send either a positive or negative response message. |
| **0x79 – 0x7D** | ISO SAE Reserved | This NRC is also reserved for the future definition. |
| **0x7E** | Sub-function Not Supported In Active Session | This NRC will return by the server if the requested sub-function Identifier is not supported in this session. basically, in UDS protocol there are 3 sessions. In each session what services and their sub-functions will support it will be decided by the OEM. If the tester will request a service with a correct subfunction for this service Identifier but that is not supported in this session, then obviously the server will send this NRC to the client. But remember there might be a possibility that the same sub-function and service Identifier will support in any other session. |
| **0x7F** | Service Not Supported In Active Session | This NRC will return by the server if the requested service identifier is not supported in this session. |
| **0x80** | ISO SAE Reserved | This NRC is also reserved for future definition. |
| **0x81** | Rpm Too High | This NRC indicates that the requested action sent by the client will not be taken because the server prerequisite condition for RPM is higher than the defined value in ECU or server. |
| **0x82** | Rpm Too Low | This NRC indicates that the requested action sent by the client will not be taken because the server prerequisite condition for RPM is lower than the defined value in ECU or server. |
| **0x83** | Engine Is Running | An NRC is necessary for actuator tests that cannot be performed while the engine is operational. This requirement is distinct from a negative response due to excessively high RPM and should be permitted. |
| **0x84** | Engine Is Not Running | An NRC is needed for actuator tests that can only be activated when the engine is in operation. This is distinct from a negative response caused by excessively low RPM and should be permitted. |
| **0x85** | Engine Run Time Too Low | This NRC signifies that the requested action will not be executed due to the server prerequisite condition, which requires the engine run time to meet a pre-programmed limit. The current engine run time is below this limit. |
| **0x86** | Temperature is Too High | This NRC informs that the requested action will not be executed due to the server’s prerequisite condition related to temperature not being met. The current temperature exceeds a pre-programmed maximum threshold. |
| **0x87** | Temperature is Too Low | This NRC signifies that the requested action will not be carried out because the server’s prerequisite condition for temperature is not satisfied. The current temperature is below a pre-programmed minimum threshold. |
| **0x88** | Vehicle Speed is Too High | This NRC communicates that the requested action will not be executed due to the server’s prerequisite condition for vehicle speed not being met. The current vehicle speed exceeds a pre-programmed maximum threshold. |
| **0x89** | Vehicle Speed is Too Low | This NRC conveys that the requested action will not be carried out because the server’s prerequisite condition for vehicle speed is not satisfied. The current vehicle speed is below a pre-programmed minimum threshold. |
| **0x8A** | Throttle/Pedal is Too High | This NRC implies that the requested action will not be executed because the server’s prerequisite condition for throttle or pedal position is not fulfilled. The current TP/APP (Throttle/Pedal Position) is above a pre-programmed maximum threshold. |
| **0X8B** | Throttle/Pedal IS Too Low | This NRC suggests that the requested action will not be carried out because the server’s prerequisite condition for throttle or pedal position is not satisfied. The current TP/APP (Throttle/Pedal Position) is below a pre-programmed minimum threshold. |
| **0X8C** | Transmission Range Is Not In Neutral | This NRC communicates that the requested action will not be executed because the server’s prerequisite condition for the vehicle being in neutral is not satisfied. The current transmission range is not in the neutral position. |
| **0x8D** | Transmission Range is Not In Gear | This NRC indicates that the requested action will not be executed because the server’s prerequisite condition for the vehicle being in gear is not fulfilled. The current transmission range is not in a gear position. |
| **0x8E** | ISO SAE Reserved | This range of values is set aside by this document for future definition. |
| **0x8F** | Brake Switch(es) Not Closed (Brake Pedal not pressed or not applied) | This NRC indicates that for safety reasons, a specific requirement is necessary for certain tests before they commence, and it must be upheld throughout the entire duration of the test. |
| **0x90** | Shifter Lever Not In Park | This NRC signifies that, due to safety considerations, a certain requirement is mandatory before the commencement of specific tests, and it must be adhered to throughout the entire duration of the test. |
| **0x91** | Torque Converter Clutch is Locked | This NRC states that the requested action will not be executed because the server’s prerequisite condition for the torque converter clutch is not satisfied. The current TCC (Torque Converter Clutch) status is above a pre-programmed limit or it is locked. |
| **0x92** | Voltage is Too High | This NRC conveys that the requested action will not be carried out because the server’s prerequisite condition for voltage at the primary pin of the server (ECU) is not fulfilled. The current voltage exceeds a pre-programmed maximum threshold. |
| **0x93** | Voltage Too Low | This NRC communicates that the requested action will not be executed because the server’s prerequisite condition for voltage at the primary pin of the server (ECU) is not satisfied. The current voltage is below a pre-programmed maximum threshold. |
| **0x94 – 0xEF** | Reserved for Specific Conditions Not Correct | This range of values is designated by this document for potential future definition or specification. |
| **0xF0 – 0xFE** | Vehicle Manufacturer Specific Conditions Not Correct | This range of values is set aside for scenarios that pertain to specific conditions determined by the vehicle manufacturer and are not considered correct. |
| **0xFF** | ISO SAE Reserved | This range of values is earmarked by this document for potential future definition or specification. |

# **4. Does UDS Protocol Work Differently in AUTOSAR Adaptive Platform?**

In Adaptive platform, the diagnostic service management (DSM) replaces the DCM module of the AUTOSAR Classic platform. However, their functionality remains essentially the same which is processing/dispatching of diagnostic services according to UDS (ISO 14229).



Another aspect to UDS implementation in Adaptive platform of AUTOSAR is that it supports only Ethernet based network technology. It implies that only DoIP transport layer (ISO 13400) can be used for communication and not CAN or CAN FD, FlexRay and others. In future releases, Adaptive platform will most probably support other transport layer of UDS other than **DoIP**.

The **DoIP** is extending for Diagnostic Over Internet Protocol. An upcoming trend in the automotive industry is to enable remote access to vehicles for a flexible interface between the diagnostic engineer and vehicle ECU. The vehicle diagnostic is getting done with a wired interface and also ECU flashing is taking a lot of time due to the low data rate.

Even if you want to do diagnose your vehicle and fix the issues but you are in a jungle or at night, so it is very difficult to get the diagnostic engineer and fix the issues. So to perform vehicle diagnostics over the air is very important nowadays. But you know till now mostly all the OEMs are using some low end diagnostic standardized protocols like KWP-2000, etc, but after the UDS protocol released that is now trending in the automotive industry.

The OEMs are using any kind of communication protocol, but they are implementing the UDS protocol for vehicle diagnostic. This is called DoCAN protocol. Like this whenever the Ethernet or Internet protocol (IP) used with UDS protocol this is called DoIP Protocol. There are obvious benefits in being able to diagnose a vehicle remotely that if a driver experiences a problem with the car can just pull over to the side and call the workshop, which may perform a diagnosis of the vehicle over the air.

**Reference Links:**

1. https://embetronicx.com/tutorials/automotive/uds-protocol/diagnostics-and-communication-management/
2. https://embedclogic.com/uds-protocol/diagnostic-communication-management/
3. https://autotechdrive.com/what-is-uds-protocol/
4. https://www.rfwireless-world.com/Terminology/UDS-DID-Table.html
5. https://piembsystech.com/autosar-dcm/
6. https://www.embeddedtutor.com/2019/09/autosar-dem-module.html
7. https://www.embeddedtutor.com/2019/09/autosar-dcm-module.html
8. https://piembsystech.com/doip-protocol/

**----------------------------------------------------------THE END----------------------------------------------------------**