On Vehicle Telematics Protocol Specification

Function Name:	Function ID:
On Vehicle Telematics Protocol Specification	FN007197

LET																								
FR																								
LET																								
FR																								

Date	LET	FR	Revisions	DR	CK

Reference:	
Prepared/Approved By: Jason Miller	
Checked By:	**Detailed By:**
Concurrence/Approval Signatures:	
Design Engineering Supervisor	
Design Engineering Manager	
Other Approvals/Concurrences (as required):	

STANDARD NOTES:
FOR CURRENT RELEASE STATUS, SEE THE WERS ENGINEERING NOTICE.
▽ CONTROL ITEM – THE ▽ ALSO IDENTIFIES CRITICAL CHARACTERISTICS DESIGNATED BY THE CROSS FUNCTIONAL TEAMS DEVELOPING THE PRODUCT. THESE, AND ADDITIONAL CRITICAL CHARACTERISTICS IDENTIFIED BY PROCESS REVIEWS, MUST APPEAR ON THE CONTROL PLANS ACCORDING TO ISO/TS 16949. THESE CONTROL PLANS REQUIRE PRODUCT ENGINEERING APPROVAL.

Frame 1 of 41	**REV**	**1.0**

Connected Vehicle and Services

On Vehicle Telematics Protocol Requirement Specification

Version 005

UNCONTROLLED COPY IF PRINTED

Version Date: March 22, 2018

FORD CONFIDENTIAL

Revision History

Date	Version	Author	Notes
Dec 23, 2014	000.1	John Schmotzer / Jason Miller	Initial Release
January 29, 2015	001	John Schmotzer / Jason Miller	Draft release to Vector
May 17, 2016	002	John Schmotzer / Jason Miller	Second official release
May 11, 2017	003	Jason Miller	Third official release (first production intent release)
October 17, 2017	003.1	Jason Miller	Draft with definition clarifications and Ethernet support
November 16, 2017	004	Jason Miller	Updated with new specification reference for Ethernet
March 22, 2018	005	Jason Miller	Minor clarifications and updated Ethernet spec number

Table of Contents

1 Document Overview

Ford's connectivity vision includes the ability to provide over the air software updates, diagnostics and prognostics, and data analytics solutions at the vehicle. To provide these features effectively a protocol to communicate data to and from the multiple ECUs needs to be developed to provide a framework for efficiently and securely transmitting the data to and from the cloud. This document describes the system use cases, architecture and requirements to provide ECUs that do not have direct connectivity to the cloud with a method of communicating to a client (tester) ECU that does have direct or indirect connectivity access.

1.1 Scope

The On Vehicle Telematics Protocol (OVTP) System Requirements Document defines the vehicle system requirements that each vehicle ECU must comply with in order to utilize the applications that are enabled by this document.
Each application outlined in this document can be implemented separately, however, in whole this document will provide the architecture that an ECU must comply with in order to access these applications.
This document does not apply to normal vehicle signaling on the vehicle's communication data link between two ECUs, or standard diagnostic message transmission between a diagnostic tester and an ECU. All message sets defined in this document shall operate in separate functional space on the vehicle network.
OVTP is currently only specified for Classical CAN, CAN FD, and Ethernet networks. Further work would be necessary to extend this to additional networks.

Table 1 — Example of OVTP relevant specifications applicable to the OSI layers

OSI seven layera	Specifications	
Application (layer 7)	OVTP Based Application Specs (e.g., OTA over OVTP, PARSED over OVTP, etc.)	On-Vehicle Telematics Protocol SRD
Presentation (layer 6)		
Session (layer 5)	On-Vehicle Telematics Protocol SRD	
Transport (layer 4)	ISO 15765-2	further standards
Network (layer 3)		further standards
Data link (layer 2)	ISO 11898-1, ISO 11898-2	further standards
Physical (layer 1)		further standards
a Seven layer according to ISO/IEC 7498-1 and ISO/IEC 10731		

1.2 Applicable Documents / References

The following documents are either referenced by this specification, or contain information that is relevant to this specification.

Table 2 — Applicable Documents / References

Reference #	Document Title	Version or Date	Document Number
[1]	Road Vehicles – Diagnostic communication over Controller Area network (DoCAN) – Part 2: Transport protocol and network layer services	2016-04-12	ISO 15765-2:2016
[2]	Road vehicles – Unified diagnostics services (UDS) – Part 2: Session layer services	2013-02-21	ISO 14229-2:2013
[3]	Global Master Reference Database (ISO 14229-1 based)	N/A	
[4]	Automotive Ethernet Link Implementation Specification	Latest Available	000601.001

1.3 Abbreviations / Acronyms

The following abbreviations and acronyms are used throughout this specification.

Table 3 — Abbreviations and Acronyms

CAN	Controlled Area Network
CAN FD	Controlled Area Network Flexible Data rate
ECG	Enhanced Central Gateway
ECU	Electronic Control Unit
FID	Function Identifier
GMRDB	Global Master Reference DataBase
IPC	Instrument Panel Cluster
OMC	OVTP Message Counter
OTA	Over The Air
OVTP	On Vehicle Telematics Protocol
PARSED	Processing and Reporting System for Efficient Data
PCM	Powertrain Control Module
SDLC	Smart Data Link Connector
SRD	System Requirements Document
SSN	Session Serial Number
TCU	Telematics Control Unit

1.4 Definitions

The following definitions apply throughout this specification.

Table 4 — Definitions

The key words "**MUST**", "**MUST** NOT", "REQUIRED", "**SHALL**", "**SHALL** NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 (see http://www.ietf.org/rfc/rfc2119.txt).	
Classical CAN	Controller area network which supports one single bit rate
CAN FD	Controller area network flexible data rate which supports two bit rates
Client	The tester ECU (normally with direct or indirect connectivity to the cloud), which is responsible for sending requests to a target ECU (server) in a request / response type OVTP application. For typical Ford vehicles implementing this specification, it is planned for this to be the Telematics Control Unit or the Enhanced Central Gateway.
Server	The target ECU that is communicating to the OVTP client. In a request / response type OVTP application, this ECU is sending the responses based upon receiving a request.
OVTP Application	A specific application (e.g., OTA) that leverages the core OVTP requirement specification to perform application specific tasks and communicate between two entities on a vehicle network(s).
OVTP Message	Refers to a complete OVTP message including header, payload, etc.
OVTP Frame	Refers to an individual OVTP frame transmitted on a given data link layer (e.g., CAN). Note that multiple OVTP frames may be necessary to send a complete OVTP message.

2 OVTP Transmit Models

OVTP based applications are generally broken into two high level models. One model is the request / response model where a client (tester) sends an application request to a server (ECU), and the server (upon receiving this request) provides an appropriate response. With this approach there is always at most one response to each request from the server. The second model is a publish / subscribe model where the server can publish messages (referred to as push messages) that are not directly triggered by a single preceding request. For this publish / subscribe model, the push messages are still always required to be sent within the context of an OVTP application session that is initiated using a request / response model. Currently, the only example of a publish / subscribe model is the PARSED push messages (see section **Error! Reference source not found.**).

2.1 Example Architecture and Use Case Showing Request / Response Model

The example in Figure 1 shows what is expected to be a typical implementation of OVTP on Ford vehicle architectures which have a TCU and an SDLC. In this example, the TCU acts as an OVTP client which sends OVTP requests on CAN to ECUs (likely on different CAN networks) through the SDLC which acts as a CAN frame based gateway. The ECUs send OVTP responses back on CAN through the SDLC to the TCU. Note that communication between the cloud and the TCU is not covered by this specification.

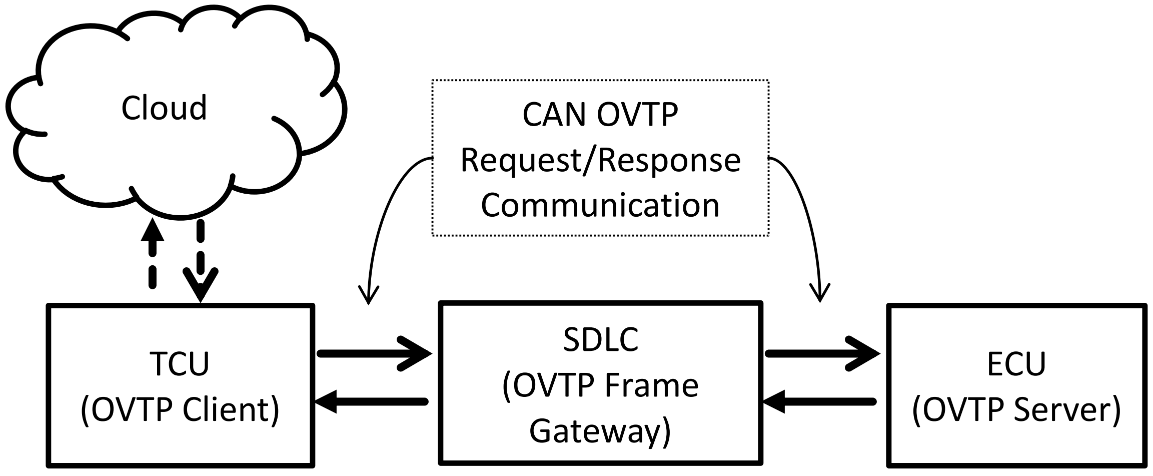
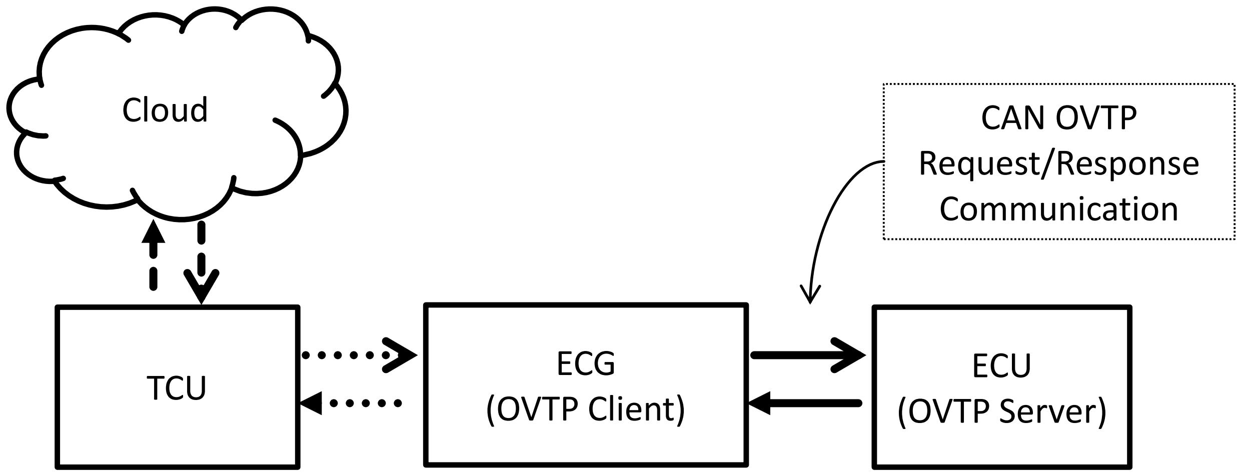

Cloud
CAN OVTP Request/Response Communication
TCU (OVTP Client)
SDLC (OVTP Frame Gateway)
ECU (OVTP Server)

Figure 1 — Example OVTP Request / Response Communication with TCU and SDLC

The example in Figure 2 shows what is expected to be a typical implementation of OVTP on Ford vehicle architectures which have a TCU and an ECG. In this example, the ECG acts as an OVTP client which sends OVTP requests on CAN directly to ECUs. The ECUs send OVTP responses back on CAN to the ECG. Note that communication between the cloud and the TCU is not covered by this specification, nor is communication between the TCU and the ECG in this example.

Figure 2 — Example OVTP Request / Response Communication with TCU and ECG



2.2 Example Architecture and Use Case Showing Publish / Subscribe Model

The example in Figure 3 shows what is expected to be a typical implementation of OVTP on Ford vehicle architectures which have a TCU and an SDLC. The TCU acts as an OVTP client to open an OVTP session in a target ECU through the SDLC as shown in Figure 1. In this example, the client directs the ECU to publish push messages which are then gatewayed through the SDLC back to the TCU.

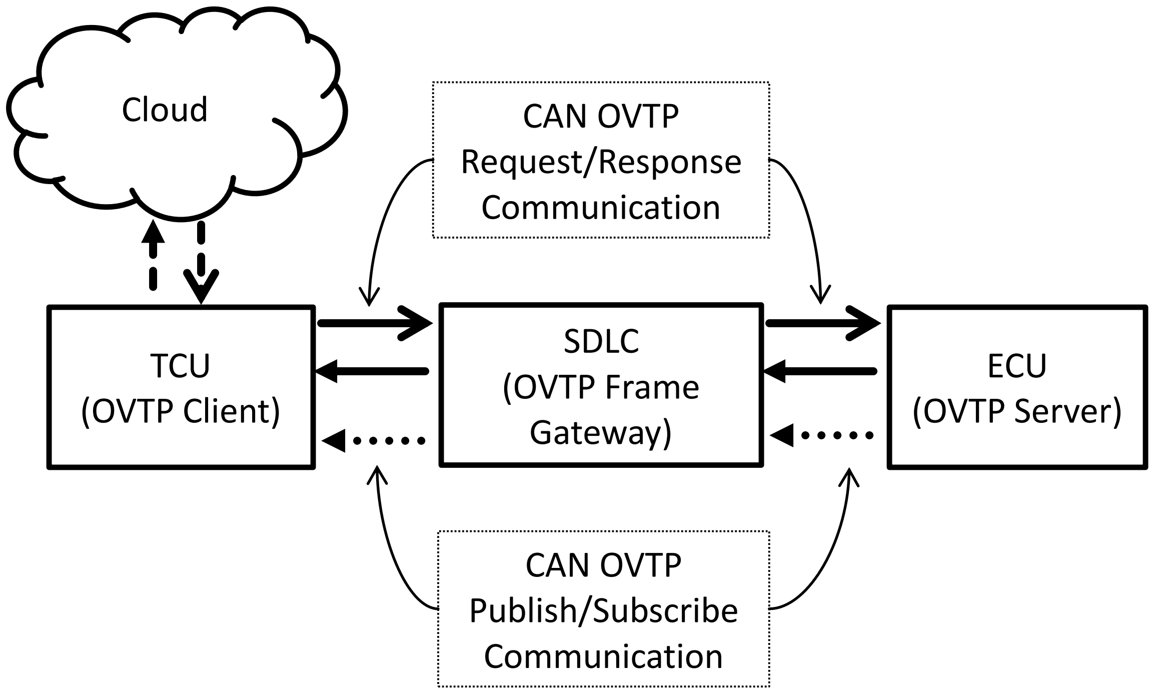

Cloud
CAN OVTP Request/Response Communication
TCU (OVTP Client)
SDLC (OVTP Frame Gateway)
ECU (OVTP Server)
CAN OVTP Publish/Subscribe Communication

Figure 3 — Example OVTP Publish / Subscribe Communication with TCU and SDLC

The example in Figure 4 shows what is expected to be a typical implementation of OVTP on Ford vehicle architectures which have a TCU and an ECG. The ECG acts as an OVTP client to open an OVTP session in a target ECU as shown in Figure 2. In this example, the client further directs the ECU to publish push messages which are then sent directly back to the ECG.

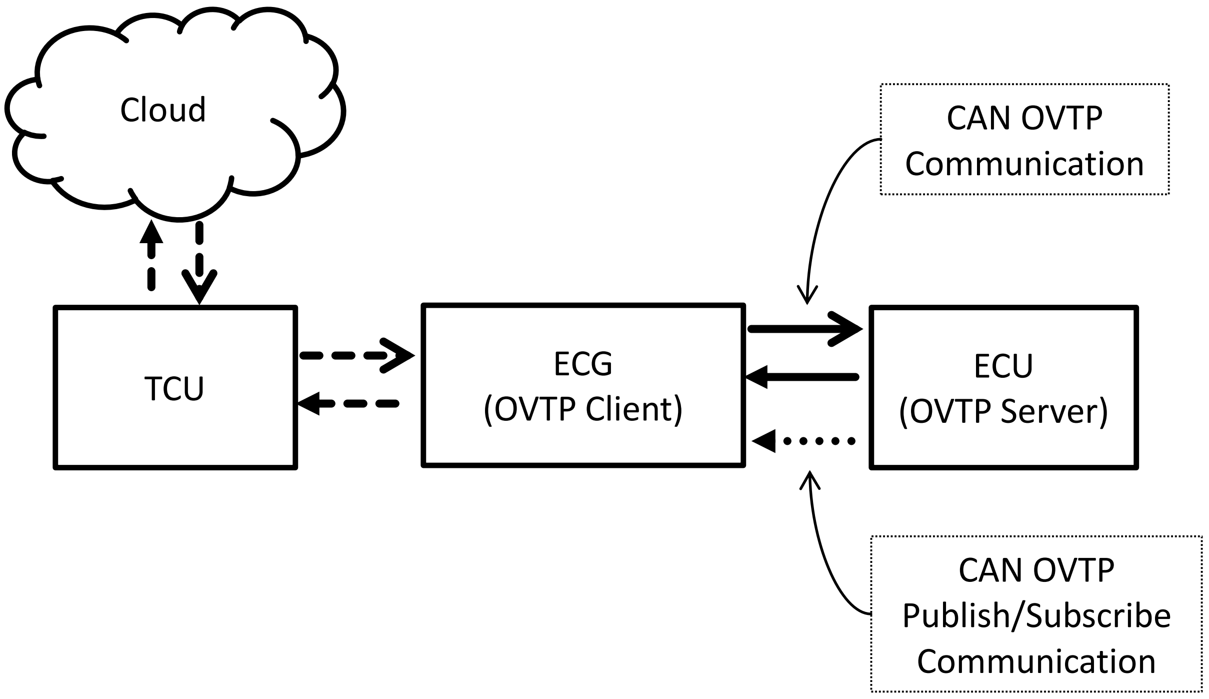


Figure 4 — Example OVTP Publish / Subscribe Communication with TCU and ECG

3 OVTP Architecture

3.1 Overview

OVTP is intended as a core building block that enables various applications to communicate with each other on Ford in-vehicle networks and is therefore broken down into various parts. These parts include the ability to specify:

- Which application the request or response is for (e.g., see section **Error! Reference source not found.**)
- The addressing of each request or response, including
    - Physical or functional request
    - Target ECU and source ECU
- Session ID handling for each application
- Standard security features for a given message (e.g., authentication)

3.2 Payload Endianness / Bit Ordering

OVTP and associated applications shall be Big Endian unless specifically stated otherwise. Additionally, all bit numbering in this specification shall always refer to bit 0 as the least significant bit.

3.3 Terms and Definitions

3.4 REQ-314705/A-OVTP Connection

This is a logical tuple of client, server, and application (i.e., Target, Source, and Application). An example connection is ECG (source) / PCM (target) / PARSED (application). A different example connection is ECG / PCM / OTA. All supported OVTP connections may need to be defined at compile time. An OVTP session is tracked per connection and this includes session timeout, connection specific TX_STmin, etc.

3.5 REQ-314706/A-OVTP Channel

The OVTP channel is used to allocate buffers and to maintain protocol processing state for actual OVTP communication to occur for a given OVTP connection. At a high level, the number of supported OVTP channels reflects the number of OVTP connections for which communication can occur simultaneously. It is anticipated that OVTP servers will have an OVTP channel for each supported application. It is anticipated that OVTP clients will have many more connections than allocated channels.
Note: PARSED and PARSED Push are two separate applications, so at least two channels (as well as two OVTP buffers) should be configured for an ECU implementing PARSED Push.

3.6 REQ-314707/A-OVTP Buffer Pool

A pool of potentially varying sized buffers allocated for OVTP communication purposes. If the given OVTP connection has an active session and communication is necessary, an available OVTP channel will be allocated and then a buffer of appropriate size is then assigned to that channel to support the transmission and/or reception.
Note: Both an available OVTP channel and an available appropriately sized buffer are required for communication to occur. Upon reception of a first frame on an active OVTP connection, if no OVTP channel or buffer pool is available a Wait Flow Control frame is sent. One single frame message will be queued per connection.

3.7 REQ-314708/A-OVTP Common Information

The following information must be capable of being included when transmitting any OVTP payload over any network.

Table 5 — Required OVTP Fields

OVTP Fields	Description	Minimum Size
Application	Used to define the application that is sending or receiving information	4 bits
Target	Used to define the ECU that will be receiving the message transmitted	10 bits
Source	Used to define the ECU that will be sending the message transmitted	10 bits

These OVTP fields are further defined in the following subsections.

3.8 REQ-314709/A-Application

The Application field defines the application that the OVTP frame is associated with. Current allowed applications are described in Table 6. Potential future OVTP applications include items such as command and control functionality, wrapped diagnostics, etc. Although not all Application field values are defined, any value within the application field (0 – 15) shall be considered a valid OVTP frame. For example, frame based CAN routers shall not care about the value within the application field when determining if a frame is an OVTP frame.

Table 6 — Application field

Application	**Value**	**OVTP Model**
Reserved for Future Use	0b0000 – 0b1000	Undefined
OTA	0b1001	Request / Response
PARSED	0b1010	Request / Response
PARSED Push	0b1011	Publish / Subscribe
Symmetric Key Distribution	0b1100	Request / Response
Reserved for Future Use	0b1100 – 0b1111	Undefined

3.9 REQ-314710/A-Application Definition

Each application using the request / response OVTP model is responsible for defining its own function ID requirements within the function ID range allocated to applications (see section **Error! Reference source not found.**). These request / response applications shall implement any function IDs within the core application independent function ID range according to this specification.

Applications using the publish / subscribe OVTP model do not implement function IDs and therefore the OVTP payload is wholly defined by the application specification. However, the OVTP publish push messages must still be enabled within the context of a related OVTP request / response application OVTP session.

OTA – This application type is intended to be used for the purposes of providing background software updates into a second memory area while an ECU application is executing. Refer to the OTA over OVTP relevant specifications for details.

PARSED – This application type is intended to be used for the purposes of defining, requesting and receiving internal data from on-board ECUs. This may consist of diagnostic information, statistical information, etc. Refer to the PARSED over OVTP relevant specifications for details.

PARSED Push – This application type is intended to push the actual data from the ECU to the PARSED OVTP client. The data is configured and requested to start being sent using the normal PARSED request/response application type. A separate application type is utilized to allow the PARSED OVTP client to continue to request

additional actions while an ECU is currently sending PARSED push data. Refer to the PARSED over OVTP relevant specifications for details.
Symmetric Key Distribution – This application type is intended to be used for the distribution and update of symmetric private keys for purposes such as CAN message authentication. Refer to the Symmetric Key Distribution over OVTP relevant specifications for details.

3.10 REQ-314711/A-Target and Source

Target and source addresses shall comply with the ECU address definitions in the GMRDB (see reference [3]). For example, a target address of 0b0000010000 shall always represent the Powertrain Control Module and a source address of 0b0010010001 shall always represent the Telematics Control Unit.
The Target field defines the target ECU (i.e., receiver) for the OVTP message. For messages originating at the OVTP client the target is defined as the ECU that is receiving the client information. In a typical request / response implementation, a request message would have the server (i.e., ECU) as the target, whereas the ECU's response message would have the client (i.e., the source in the request message) as the target.
The Source field defines the originating ECU (i.e., transmitter) for the OVTP message. For messages originating at the OVTP client the source is defined as the Client that is sending the information to the Server. In a typical request / response implementation, a request message would have the client's own address as the source, whereas the ECU's response message would have the ECU's own address as the source.
Depending on the application needs, OVTP request / response applications may require support for functional addressing. Functionally addressed requests shall utilize a target address of 0x3FF as defined in the GMRDB. A functionally address target address is only allowed in OVTP request frames. An OVTP response to a functionally addressed request shall always use the OVTP server's address. The functionally addressed address of 0x3FF is never allowed to be used as a source address in an OVTP message.
If functional addressing is supported for an OVTP application, it shall be supported independent of function ID (i.e., for all function IDs supported by the application). If functional addressing is used, it is the responsibility of the client to ensure it has adequate resources (e.g., OVTP connections, buffers, etc.) to handle all expected responses.
Any transmitted publish / subscribe push messages shall always use the address of the last client which successfully sent an openSession (see section **Error! Reference source not found.**) message to the server. For request / response models, the response message shall always use the client's address from the corresponding request message.

3.11 REQ-314712/A-OVTP on Classical CAN or CAN FD

Physical layer requirements for both Classical CAN and CAN FD are not described within this specification as those requirements are specified by Ford Core Network Communications area. From an OVTP perspective, an ECU that is on a Classical CAN only network (i.e., one that is not CAN FD capable due to one or more ECUs not being CAN FD capable), the ECU shall only be required to implement OVTP support on Classical CAN. If this same ECU is CAN FD capable, it may additionally implement OVTP support on CAN FD to allow a future transition to a fully CAN FD capable network. An ECU that is on a CAN FD capable network (i.e., one where all ECUs on the network support CAN FD and therefore CAN FD frames may be sent and acknowledged), the ECU shall support OVTP on both Classical CAN and CAN FD.
If an OVTP server application supports both Classical CAN and CAN FD, then any request received on Classical CAN shall always result in a response sent on Classical CAN. Similarly, any request received on CAN FD shall always result in a response sent on CAN FD. For any publish / subscribe models, the OVTP application publish messages shall always be sent on the same CAN variant for which the corresponding OVTP application request / response session was opened with. For example, if an OVTP PARSED session was started on CAN FD, then the OVTP PARSED push messages shall be transmitted on CAN FD.

3.12 Network / Transport Layer

All OVTP messages shall be fully compliant with the ISO 15765-2 published 2016-04-12 with the additional details described in the following sub-sections.

3.12.1 REQ-314714/A-ISO 15765-2 Buffer Size

The ISO 15765-2 buffer must be sized to at least support the maximum possible expected receive message based upon the specific OVTP application requirements (e.g., if an OTA over OVTP server needs to receive 1024 byte request messages to support programming the buffer allocated to OTA over OVTP must be sized to at least 1024 bytes). However, the transport layer buffer size for any OVTP application shall not exceed 4095 bytes. Note that this shall be interpreted to mean that the FirstFrame data length escape sequence of all 0s shall not be supported.

3.12.2 REQ-314715/A-Flow control parameters

The following sections detail the valid usage of the flow control parameters for main nodes as defined in ISO 15765-2.

3.12.3 REQ-314716/A-FlowStatus (FS) parameter

The valid FlowStatus parameters are ContinueToSend (0_H), Wait (1_H), and Overflow (2_H). Only the client (tester) may transmit a flow control frame with a FlowStatus of Wait (1_H). All OVTP implementations shall support reception of a flow control frame with a FlowStatus of Wait (1_H).

3.12.4 REQ-314717/A-BlockSize (BS) parameter

All flow control frames transmitted from a client or ECU shall utilize a BlockSize of 00_H.

3.12.5 REQ-314718/A-SeparationTime (STmin) parameter

The STmin parameter transmitted by the client shall use a value based upon the needs of the specific OVTP application. The STmin parameter in flow control frames transmitted by an ECU is specified by the implementer based upon the requirements specified by the specific OVTP application.

3.12.6 REQ-314719/A-Timing parameters

The network layer timing parameter values, timeout and performance requirement values, shall be as detailed in **Table 7** below. These values shall be used by the OVTP client and the ECU for all messages. Refer to ISO 15765-2 for definitions of each parameter.

Table 7 — ISO 15765-2 Timing Parameter Values

Parameter		
Symbol	Value, hex	
	Timeout, ms See note 1	Performance requirement, ms See note 2
N_As/N_Ar	1000	-
N_Bs	1000	-
N_Br		(N_Br + N_Ar) < 250
N_Cs	-	(N_Cs + N_As) < (250 or (Tx_STmin + 50)) whichever is greater

N_Cr	1000 See note 3	-

Notes in Table 7

1. Actual timeout values shall be implemented at the upper end of the timeout range. For example, if the earliest the timeout can occur is 1000ms and the latest the timeout can occur is 1500ms (see ISO 15675-2 for tolerances), then for an ECU which supports an 8 ms window the guaranteed timeout range should be 1488ms – 1496ms.
2. A given OVTP application may choose to enforce stricter performance requirements.
3. An OVTP client shall increase its default N_Cr timeout if necessary to account for any planned Tx_STmin values (see **Error! Reference source not found.**) it configures in a server to prevent the N_Cr timeout from occurring during normal operation.

3.12.7 REQ-314720/A-Maximum number of FC.Wait frame transmission (N_WFTmax)

The value of the N_WFTmax parameter (defined in ISO 15765-2) for an OVTP application shall be 600.

3.12.8 REQ-314721/A-Half and Full Duplex

Only half-duplex shall be supported for clients and servers on OVTP.

3.12.9 REQ-314722/A-Bandwidth Utilization and Time Between OTVP Frames

In many cases, a given application using OVTP to transmit messages may only be granted a limited portion of the total available CAN bandwidth. Therefore it is important to be able to define the shortest time between any two OVTP frames for a given OVTP connection. To help ensure the allocated bandwidth is not exceeded, a run-time configurable parameter called Tx_STmin shall be introduced, which is configurable for a given OVTP session.
Tx_STmin effectively limits the fastest any two frames for a given OVTP connection can be transmitted in 2 ways. The first way is by controlling the time between any two consecutive frames during a segmented message. N_Cs is a parameter defined by ISO 15765-2 as the actual attempted time between the transmission of two consecutive frames. By default, its lower limit (i.e., the shortest time the transport layer can wait) is driven by the STmin value received as part of the flow control frame. Alternatively, the upper limit the transport layer can wait is driven by the N_Cr timeout in the receiver and any performance requirements (i.e., N_As+N_Cs). To meet potential bus utilization targets, the actual transmission time between two consecutive frames may be required to be longer than the fastest time at which the receiver can accept the frames (i.e., the received STmin). With the introduction of this Tx_STmin parameter, the new lower limit of the N_Cs (i.e., the minimum time the transport layer must wait between consecutive frames) shall be the maximum of the received STmin in the flow control and the locally configured Tx_STmin.
For example, if the Tx_STmin is 20ms and the receiver sends a STmin of 5ms in its FlowControl frame, the sender shall ensure that consecutive frames are not sent any faster than 20ms apart. Alternatively, if the Tx_STmin is 20ms and the receiver sends a STmin of 25ms in its FlowControl frame, the sender shall ensure that consecutive frames are not sent any faster than 25ms apart.
The second way Tx_STmin limits the time between any two frames for a given OVTP connection applies to both request / response messages and publish / subscribe push messages. In this case, the Tx_STmin shall force a minimum time between the end of one transmitted message and the start of another for a given OVTP application. For example, in a publish / subscribe model, this protects for the case that published push messages may be event based and several small (e.g., single frame) publish push messages could be queued up at the same time. In a request / response model, this parameter would delay a new request from a client from being transmitted to the same server if Tx_STmin has not expired since the previous request was transmitted
The Tx_STmin value shall be run-time configurable with a resolution of 10ms and a range of 0 ms to 65535 ms.
The expectation is that the client for a given OVTP application will control the worst case overall bandwidth utilization by locally controlling its own Tx_STmin, and setting the Tx_STmin in each server during the openSession request (see section **Error! Reference source not found.**).

3.13 REQ-314723/A-Data Link Layer

Only 29-bit CAN IDs shall be utilized for OVTP messages on both Classical CAN and CAN FD. All messages will use Unacknowledged Segmented Data Transfer (USDT) according to ISO 15765-2 using normal addressing. The required mapping of the 29-bit CAN ID is described in the following sub-sections

3.14 CAN ID Details

The applications that are serviced using OVTP require information that can properly route the information provided in the payload. OVTP on CAN accomplishes this by requiring a specific mapping of the 29-bit CAN IDs. The required 29-bit CAN ID utilization for OVTP is described in **Table 8** with definitions for each field described in **Table 9**. As described in the following subsections, an OVTP frame can always be identified by a value of 0b11011 in bits 28 – 24 of a 29-bit CAN frame.

Table 8 — 29-bit CAN ID for OVTP

Frame Type	29 bit CAN identifier bit position				
	28 – 26	25 – 24	23 – 20	19 – 10	9 – 0
OVTP Frame	Priority 0b110	<Reserved> 0b11	Application	N_TA	N_SA

Table 9 — OVTP 29-bit CAN ID Field Definitions

Header Parameter	Description	Bit Allocation
Priority	Used to define the priority of the message relative to other frames on the CAN bus.	3 bits
<Reserved>	Reserved bits for future use	2 bits
Application	See Table 5.	4 bits
Target	See Table 5.	10 bits
Source	See Table 5.	10 bits

3.14.1 REQ-314725/A-Priority

The Priority field defines the priority of the messages relative to all other frames on the CAN network. For OVTP, this value shall always be a value of 6 (0b110). If any other value is present within this field it shall not be considered a valid OVTP frame.

3.14.2 REQ-314726/A-<Reserved>

The <Reserved> field defines a section of the 29-bit CAN ID reserved for future development that is not in scope for the current implementation. For all current implementations of OVTP, this value shall always be a value of 3 (0b11). If any other value is present within this field it shall not be considered a valid OVTP frame.

3.14.3 REQ-314727/A-Application

See section **Error! Reference source not found.**

3.14.4 REQ-314728/A-Target and Source

See section **Error! Reference source not found.**.

Functional requests must always fit into a single frame as defined by ISO 15765-2. Therefore, while all function IDs are supported by the application layer if received functionally, not all requests may be capable of being sent functionally over ISO 15765-2 (e.g., they may not fit into a single frame).

3.14.5 Example OVTP CAN ID Values

Table 10 — Example CAN IDs on OVTP

Sender	Receiver	Application	29-bit CAN ID
TCU (0x91)	PCM (0x10)	PARSED (0b1010)	0x1BA04091
PCM (0x10)	TCU (0x91)	PARSED (0b1010)	0x1BA24410
PCM (0x10)	TCU (0x91)	PARSED Push (0b1011)	0x1BB24410
TCU (0x91)	IPC (0x60)	OTA (0b1001)	0x1B918091
IPC (0x60)	TCU (0x91)	OTA (0b1001)	0x1B924460
TCU (0x91)	Functional (0x3FF)	OTA (0b1001)	0x1B9FFC91
TCU (0x91)	Functional (0x3FF)	PARSED (0b1010)	0x1BAFFC91

3.15 REQ-314730/A-CAN frame data length

For both Classical CAN and CAN FD the CAN frame data padding option shall be implemented so that every transmitted OVTP CAN frame shall have a minimum data length code of eight (8). Any OVTP CAN frame with a data length code less than eight shall be considered invalidly formatted as described in ISO 15765-2 and completely ignored by the receiver. Any unused bytes in every OVTP CAN frame transmitted shall be padded with 0xCC to minimize stuff-bit insertions. However, a receiver shall never reject an OVTP frame based upon pad byte values.

3.16 REQ-314731/A-OVTP on Ethernet

Physical layer requirements for Ethernet are not described within this specification as those requirements are specified by Ford Core Network Communications area.
The OVTP message payload (see section **Error! Reference source not found.**) shall be sent as described in reference [4]. Reference [4] describes how the OVTP Application, Target, and Source fields from section **Error! Reference source not found.** are mapped into the Ethernet frames.

4 Timing Parameter Definition

4.1 REQ-314733/A-General application timing considerations

This section applies only to OVTP applications implementing the request / response model. The timing parameters specified in this specification are intended to be equivalent to their counterparts used for diagnostics as defined in ISO 14229-2 (e.g., $F2_{Server}$ is functionally equivalent to $P2_{Server}$). Additionally, refer to ISO 14229-2 and ISO 15765-2 for the transport network layer PDUs and service primitives (e.g., SOM.ind).

4.2 REQ-314734/A-Preparedness for Requests

Each OVTP application shall specify when a server must be required to accept OVTP request messages including items such as power-up times, ignition status, etc. If not otherwise specified by a given OVTP application specification, the OVTP server shall always be able to receive, interpret, and execute requests and send responses to every correct request when the ECU is awake and operating its application. The exception to this is that the server is allowed a maximum delay of 1000ms following a power up or a reset in which it is not required to respond to OVTP requests.
If a particular OVTP application receives an additional OVTP request (physically or functionally addressed) before the application has completed processing the previous request, it shall completely ignore the additional OVTP application request. This is true if the additional request is from the same or a different client as the request currently being processed.
A functionally addressed and validly formatted requestSessionStatus with suppressResponseIndication set to “suppress positive response” is not allowed to “block” the application (see section **Error! Reference source not found.**) and therefore can never be considered a previous request that is still being processed.

4.3 REQ-314735/A-Server

A server uses an application specific timer ($F2_{Server}$) implementation which is triggered (started, reloaded, and stopped) by the T_Data service primitive interface (T_Data.req, T_Data.con, T_DataSOM.ind, T_Data.ind).
The $F2_{Server}$ application timer is loaded with a $F2_{Server\_max}$/$F2^*_{Server\_max}$ parameter value. Both parameters and values are specified in this part of OVTP (see definition in Table 11 and parameter value in Table 12).

The timing parameter F4 is the required performance time between the reception of a request and the start of transmission of the final response. A final response is a positive response or a negative response other than negative response code 0x78 (requestCorrectlyReceived-ResponsePending). F4 is a performance requirement that is specified for a given function ID or data. $F4_{Server\_max}$ is the maximum value of F4. If $F4_{Server\_max}$ is the same as $F2_{Server\_max}$, this means that a negative response with negative response code 0x78 is not allowed for that function ID.
These requirements are applicable only to function IDs that are supported by the server/ECU. Unsupported function IDs shall always utilize a $F4_{Server\_max}$ value equal to $F2_{Server\_max}$ (i.e., NRC 0x78 not allowed).

4.4 REQ-314736/A-Client

A client uses a single application timer ($F2_{Client}$) implementation which is triggered (started, reloaded, and stopped) by the T_Data service primitive interface (T_Data.req, T_Data.con, T_DataSOM.ind, T_Data.ind).
The $F2_{Client}$ application timer is always loaded with a $F2_{Client\_max}$/$F2^*_{Client\_max}$.
The $F2_{Client}$ application timer is started, whenever the client application layer receives a T_Data.con service primitive and is loaded with $F2_{Client\_max}$ parameter value.
The $F2_{Client}$ application timer is stopped, either when the T_DataSOM.ind or the T_Data.ind service primitive is received by the application.
The client application verifies the correct application timing by comparing its actual $F2_{Client}$ application timer with the $F2_{Client\_max}$ parameter value. If T_DataSOM.ind or T_Data.ind is received while $F2_{Client}$ is smaller or equal to $F2_{Client\_max}$ the timing fulfils the requirements established by this standard. If no .ind is received while $F2_{Client}$ is smaller or equal to

$F2_{Client\_max}$ an error condition is detected. This shall be flagged to the application layer with the parameters included in either T_DataSOM.ind or the T_Data.ind service primitive.
All parameters and values are specified (see definition in Table 11 and parameter values in Table 12).

Table 11 — Message timing parameter definitions

Timing Parameter	Description	Type
$\Delta F2$	The $\Delta F2$ parameter is defined to be the worst case vehicle network design-dependent message transmission delay such as delays introduced by gateways and bus-load dependent arbitration.	Performance requirement
$F2_{Server}$	Performance requirement for the server to start with the response message after the reception of a request message (indicated via T_Data.ind).	Performance requirement
$F2_{Client}$	Timeout for the client to wait after the successful transmission of a request message (indicated via T_Data.con) for the start of incoming response messages (if supported via T_DataSOM.ind) or completion of the response message (indicated by T_Data.ind).	Timer reload value
$F2*_{Server}$	Performance requirement for the server to start with the response message after the transmission of a negative response message (indicated via T_Data.con) with negative response code 0x78 (enhanced response timing).	Performance requirement
$F2*_{Client}$	Enhanced timeout for the client to wait after the reception of a negative response message with negative response code 0x78 (indicated via T_Data.ind) for the start of incoming response messages (indicated via T_DataSOM.ind of a multi-frame message or T_Data.ind of a SingleFrame message).	Timer reload value
$F3_{Client\_Phys}$	Minimum time for the client to wait after the successful transmission of a physically addressed request message (indicated via T_Data.con) with no response required before it can transmit the next physically addressed request message.	Timer reload value
$F3_{Client\_Func}$	Minimum time for the client to wait after the successful transmission of a functionally addressed request message (indicated via T_Data.con) before it can transmit the next functionally addressed request message in case no response is required or the requested data is only supported by a subset of the functionally addressed.	Timer reload value
$F4_{Server}$	This is the time between the reception of a request (T_Data.indication) and the start of the transmission of the final response (T_Data.request) at the server side.	Performance requirement

Table 12 — Recommended message timing parameter value definitions

Timing Parameter	Minimum	Maximum
$\Delta F2$	0 ms	100ms
$F2_{Server}$	0 ms	200ms
$F2_{Client}$	$F2_{Server\_max} + \Delta F2_{max}$	--- a)
$F2*_{Server}$	0 ms	10000ms
$F2*_{Client}$	$F2*_{Server\_max} + \Delta F2_{response}$	--- c)
$F3_{Client\_Phys}$	$F2_{Server\_max} + \Delta F2_{max}$	--- d)
$F3_{Client\_Func}$	$F2_{Server\_max} + \Delta F2_{max}$	--- d)
$F4_{Server}$	$F2_{Server}$	OVTP Function ID Dependent

a The maximum time a client waits for a response message is at the discretion of the client, provided that $F2_{Client}$ is greater than the specified minimum value of $F2_{Client}$.
b During the enhanced response timing, the minimum time between the transmission of consecutive negative messages (each with negative response code 0x78) shall be 0.3 * $F2*_{Server\_max}$, in order to avoid flooding the data link with unnecessary negative response code 0x78 messages.
c The maximum value that a client uses for $F2*_{Client}$ is at the discretion of the client, provided it is greater than the specified minimum value of $F2*_{Client}$.
d The maximum time a client waits until it transmits the next request message is at the discretion of the client

The parameter values in Table 12 are intended to be the default values if not otherwise specified by the specific OVTP application. A given OVTP application may require that the timer values or performance values may be smaller or greater depending upon the needs of the application.

4.5 REQ-314737/A-Session Timeout

When a session timeout is requested by the client (see function openSession), the server shall implement an $S3_{Server}$ timeout and automatically terminate the session when this timeout expires.
The $S3_{Client}$ parameter shall be used by the client to maintain open connections with servers that have timed sessions.

Table 13 — Session timing parameter definitions

Timing Parameter	Description	Type	Recommended Reload ms	Timeout ms
$S3_{Client}$	Time between request messages transmitted by the client to keep an OVTP session active in a server. The $S3_{Client}$ includes the travel time of the message on the network (gateway delays etc.). Only applies if a sessionTimeout was specified when the session was opened.	Timer Reload Value	0.4 * sessionTimeout requested when session was opened	< $S3_{Server}$
$S3_{Server}$	Time for the server to keep an OVTP session active when not receiving OVTP request messages. Only applies if a sessionTimeout was specified when the session was opened. The tolerance of $S3_{Server}$ is -0m, +200ms.	Timer Reload Value	N/A	sessionTimeout requested when session was opened

Table 13 defines the conditions for the client and the server to start/restart its $S3_{Client}$/$S3_{Server}$ timer. For the client a periodically transmitted functionally addressed requestSessionStatus (0x03) request message shall be distinguished from a sequentially transmitted physically addressed requestSessionStatus (0x03) request message, which is only transmitted in case of the absence of any other OVTP request message. For the server, the functionally addressed keep-alive-logic of a functionally addressed valid requestSessionStatus with suppressResponseIndication set to “suppress positive response” has to be processed by bypass logic. It is up to the server to make sure that this specific message cannot “block” the server’s application layer and that an immediately following addressed message can be processed.
Furthermore, Table 13 shows that the $S3_{Server}$ timer handling is based on the transport/network layer service primitives, which means that the $S3_{Server}$ timer is also restarted upon the reception of an OVTP request message that is not supported by the server. The additional timer resource requirements given in Table 14 shall apply for the client and the server.

Table 14 — Session layer timing start/stop conditions for the client and the server

Timing parameter	Action	Physical and functional communication, using functionally addressed, periodically transmitted requestSessionStatus message	Physical communication only, using a physically addressed, sequentially transmitted requestSessionStatus message
$S3_{Client}$	Initial start	T_Data.con that indicates the completion of the openSession (0x01) request message.	
	Subsequent start	T_Data.con that indicates the completion of the functionally addressed requestSessionStatus (0x03) request message, which is transmitted each time the $S3_{Client}$ timer times out.	T_Data.con that indicates the completion of any request message in case no response is required.
			T_Data.con that indicates an error during the transmission of either a single-frame or multi-frame request message.
			T_Data.ind that indicates the reception of any response message in case a response is required.
			T_Data.ind that indicates an error during the reception of a multi-frame response message.
$S3_{Server}$	Initial start	T_Data.con that indicates the completion of the transmission of an openSession positive response message.	
	Subsequent stop	T_DataSOM.ind that indicates the start of a multi-frame request message or T_Data.ind that indicates the reception of any SingleFrame request message.	
	Subsequent start	T_Data.con that indicates the completion of any response message that concludes a function ID execution (final response message) in case a response message is required/allowed to be transmitted (this includes positive and negative response messages). A negative response with negative response code 0x78 does not restart the $S3_{Server}$ timer.	
		Completion of the requested action (function ID conclusion) in case no response message (positive and negative) is required/allowed.	
		T_Data.ind that indicates an error during the reception of a multi-frame request message.	

5 Generic OVTP Message Structure

5.1 REQ-314739/A-General

From an OSI application layer perspective, all OVTP messages start with a one byte OVTP Header. Depending upon the OVTP header details, the message will consist of an optional 2 byte Session Serial Number, an optional OVTP Message Counter, an OVTP Application Data Payload, and an optional Authorization Bytes. The requirements for the structure of all OVTP messages are described in Table 15 and Table 16 below. Note that the OVTP message structure applies to all OVTP models (i.e., both request / response and publish / subscribe).

Table 15 — OVTP Message Structure

Byte #	Payload	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Byte 0	OVTP Header	**Protocol Version**			**Cntr?**	**Crypto Type**			**SSN?**
Byte 1	SSN [0]	**Session Serial Number [0]**							
Byte 2	SSN [1]	**Session Serial Number [1]**							
Byte 3	OMC	**OVTP Message Counter**							
Byte 4	A_Data	**OVTP Application Data (A_Data)**							
:	:								
:	:								
:	:								
:	:								
:	:								
:	:								
:	:								
:	:								
:	:								
:	:								
Byte n - Auth Len	A_Data								
Byte n - (Auth Len - 1)	Auth Bytes	**Authentication Bytes [0]** **…** **Authentication Bytes [Auth Len-1]**							
:	:								
:	:								
:	:								
:	:								
:	:								
:	:								
Byte n	Auth Bytes								

Table 16 — OVTP Header Definition

Header Field	Definition
Protocol Version	This is the version of the core OVTP protocol. It ensures a common understanding between the communicating parties. All ECUs implementing to this specification shall use a Protocol Version value of 2 (0b010).
Cntr?	A single bit which indicates whether or not the OVTP Message Counter field is used (and therefore is present in the message). A value of 1 indicates that the one byte OVTP Message Counter is present and a value of 0 indicates that the one byte OVTP Message Counter is not present in the message. Each OVTP application must specify if the OVTP Message Counter is used.
Crypto Type	This indicates the type of cryptography, if any, used in this message. A value of 0 (0b000) shall always indicate that no cryptography is used and therefore no Authentication Bytes are included in the message. For all other values, the required Authentication Bytes are included in the message. Each OVTP application must specify if Crypto Type is required to be used for specific requests, responses, or push messages. Note that for the initial OVTP release, no Crypto Types are defined by this specification. Future Crypto Types may be defined either within this specification or a separate future specification.
SSN?	A single bit which indicates whether or not the Session Serial Number field is used (and therefore is present in the message). A value of 1 indicates that the two byte Session Serial Number is present and a value of 0 indicates that the two byte Session Serial Number is not present in the message. Each OVTP application must specify when the Session Serial Number is used.
Session Serial Number	Two byte Session Serial Number intended to assist in ensuring authenticity of a given OVTP message.
OVTP Message Counter	Four byte incrementing counter value (initialized at 0x00000000 when delivered to Ford) intended to assist in keeping client and server communication in sync and assist in detection of dropped messages. Only the least significant byte is transmitted within the OVTP message itself. This counter shall be stored in non-volatile memory and shall not be reset. An example planned typical current use case for the counter is to be incremented with each transmitted PARSED push message on a given OVTP connection in order to allow detection of dropped messages. Future usage for request / response models in conjunction with Crypto Types is envisioned to ensure more secure communication. However, the associated error handling and key distribution necessary to implement this effectively is not yet defined so this is not part of the initial OVTP implementation.
OVTP Application Data	True application payload of the message that will be passed to the application leveraging OVTP. This field shall always be present and therefore must contain at least one byte.
Authentication Bytes	The bytes used for the cryptographic authorization mechanism used (if any). The meaning, usage and length of these bytes are controlled by the Crypto Type field.

Table 17 below demonstrates example OVTP message payloads based upon different OVTP header fields and OVTP application data.

Table 17 — OVTP Header and Message Examples

Protocol Version	Cntr?	Crypto Type	SSN?	Cntr Value (hex)	SSN Value (hex)	A_Data Length	A_Data (hex)	OVTP Msg Length	OVTP Message Payload (hex)
002	0	000	1	*Not Present*	ABCD	3	01 00 00	6	41 AB CD 01 00 00
002	1	000	1	05	ABCD	1	81	5	51 AB CD 05 81
002	1	000	0	F3	*Not Present*	3	18 19 1A	5	50 F3 18 19 1A

5.2 REQ-314740/A-Valid OVTP Messages and Error Handling

When an OVTP message is received, it must always be assumed the first OVTP header byte is accurate as this must be used to determine where A_Data starts. If the length of a received OVTP message is not valid based upon the OVTP header byte, then the OVTP message is not valid and shall not be passed further to the OVTP application layer. For example, if the “Cntr?” value is 1 and the “SSN?” is 1, then the OVTP message must have a minimum length of 5 (1 byte OVTP header byte, 2 byte Session Serial Number, 1 byte OVTP Message Counter, and 1 byte A_Data). Using this example, a received OVTP message of length 4 or less would not be passed on to the OVTP application which means no response (positive or negative) would be elicited if the received message was on a request / response model.

Each OVTP application must specify the allowed values of Cntr?, SSN?, and CryptoType for each supported A_Data request (on a request / response model) and for each A_Data payload type for push message (on a publish / response model). If the received OVTP message does not match the allowed values for any of these OVTP header byte fields, the OVTP message shall be considered invalid and shall not be passed further to the OVTP application layer. This means that no response (positive or negative) would be elicited if the received message was on a request / response model. The default for unsupported A_Data request function IDs for a given OVTP application shall assume expected OVTP header values of Cntr?, SSN?, and CryptoType to match the configured expected values for the openSession function ID for that OVTP application.

If the OVTP header byte fields do match the allowed values for the specific A_Data payload (function ID or otherwise), and the length is minimally valid, then the received OVTP message shall be passed on for further inspection (e.g., session serial number matches expected value, authentication bytes are valid, function ID supported, etc.).

6 Function ID Protocol (Request / Response Models)

6.1 REQ-314742/A-General

For OVTP request / response models, the application layer protocol shall always be a confirmed message transmission, meaning that for each OVTP request sent from the client, there shall be one OVTP response sent from the server. The only exception from this rule shall be a few cases when functional addressing is used or the OVTP request specifies that no response shall be generated.

Function IDs only apply to OVTP applications implementing the request / response model. These application functions provide access to ECU centric information and tasks relating to that application. An OVTP application function will provide ECU specific data back to the client as well as allow the client to execute commands on the ECU that are application specific. For request / response OVTP models, the first byte of the OVTP Application Data (A_Data) shall always be the function ID. For publish / subscribe OVTP models, the entire OVTP Application Data (A_Data) are defined by the relevant application implementation specification.

6.2 REQ-314743/A-Function definition conventions

For a given function request and response definition, the presence of each parameter is described by one of the convention (Cvt) values as defined in Table 18.

Table 18 — Function ID parameter conventions

Type	Name	Description
M	Mandatory	The parameter has to be present.
C	Conditional	The parameter can be present, based on certain criteria (e.g., other parameters within the function definition).
U	User option	The parameter may or may not be present, depending on dynamic usage by the user.

6.3 REQ-314744/A-Function ID Ranges

The function IDs shall be defined according to Table 19. The core application independent function IDs shall be defined by this specification. The application specific function IDs shall be specified by the relevant OVTP application specific implementation document. For example, function ID 0x01 (openSession) shall always have the same core meaning as it is within the core application independent range. However, function ID 0x10 may have a different meaning from one OVTP application to another.

Table 19 — General Function ID Ranges for OVTP Request/Response Model

Function ID Range	Description
0x00 - 0x0F	Core application independent request function IDs
0x10 – 0x7E	OVTP application specific request function IDs
0x7F	Generic (application independent) negative response function ID
0x80 - 0x8F	Core application independent function positive responses
0x90 – 0xFE	OVTP application specific positive responses
0xFF	Reserved

There is a one-to-one correspondence between function identifiers for request messages and function identifiers for response messages, with bit 7 of the function ID indicating which. All request messages have a function ID with bit 7 = 0 and their corresponding positive responses shall always have the same function ID value but with bit 7 = 1.

All core application independent request functions defined in this document shall always have $F4_{Server\_max}$ equal to $F2_{Server\_max}$.

6.4 REQ-314745/A-OVTP Session Handling

An OVTP application session is used to provide a context for accessing necessary OVTP application functions as well as providing potential security benefits. The OVTP application session is used in the request / response model (i.e., client-server based system) to open and close access to ECU functions. The client uses the session to inform a ECU that the specific OVTP based application is planning to request some functionality. The server uses the session to prepare for reception of requests from the client and initiate any functionality the OVTP based application may require. A specific OVTP application on a server shall at most have a single OVTP session active, regardless of client.

The core application independent OVTP function IDs defined within this specification provide the client the capability to manage an OVTP application session. This includes the capability to open a session, close a session, and request the status of a session. A given OVTP server shall always have at most one OVTP session active for a given OVTP application (e.g., only a single OVTP OTA application session active). This means that an OVTP server either has a session active for a given OVTP application or it does not. Session handling between OVTP applications is not related (i.e., an OVTP OTA application session can be active independently from an OVTP PARSED application session).

All function IDs (application specific and not) shall always be performed within the context of an active OVTP session. The only exceptions to this are the openSession function and the requestSessionStatus function. This means the openSession function must precede all other OVTP function ID requests in order for the ECU to positively accept the request (with the exception of requestSessionStatus). In other words, for an application specific function ID to provide a positive response, it must be received within an open and active OVTP session. Refer to Figure 5 for a high level representation of an OVTP session flow.

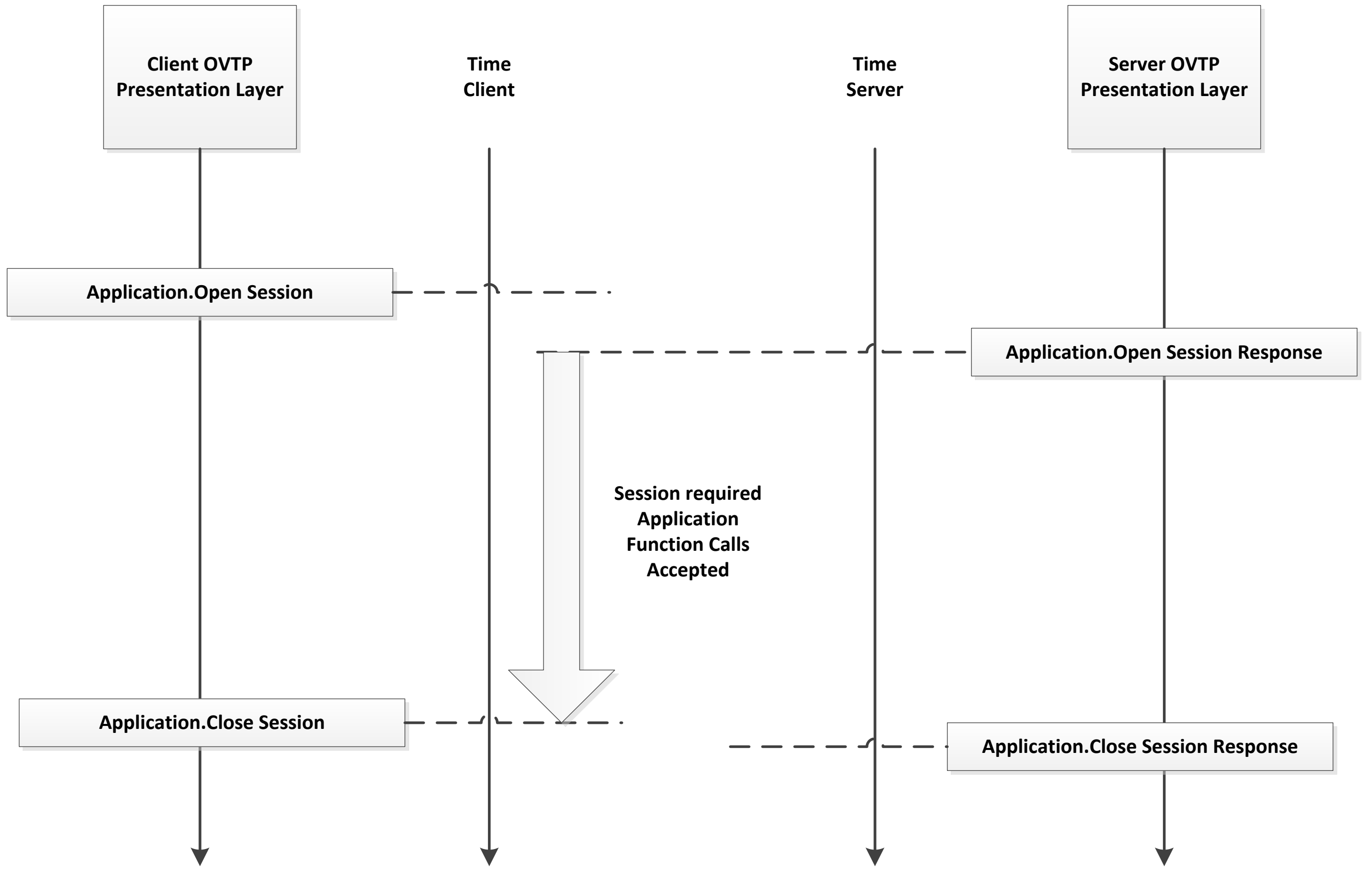

Client OVTP Presentation Layer
Time Client
Time Server
Server OVTP Presentation Layer
Application.Open Session
Application.Open Session Response
Session required Application Function Calls Accepted
Application.Close Session
Application.Close Session Response

Figure 5 — Application Session Flow

If an application specific function ID is received outside of an open OVTP application session, the server shall always respond with an NRC 0x7F (noActiveSession). The only exception to this is that a negative response with NRC 0x7F (noActiveSession) shall never be transmitted when functional addressing was used for the request message in order to prevent flooding the bus with unnecessary responses.

Table 20 — Server Session Error Handling

sessionSerial-Number Received	Server Status	Behavior
Don't care	No session active	Client sends validly formatted openSession request Server sends positive response (assuming conditions are correct) and activates new session
Don't care	No session active	Client sends any physically addressed request (other than openSession or requestSessionStatus) Server sends NRC of 0x7F (noActiveSession)
Don't care	No session active	Client sends any functionally addressed request (other than openSession or requestSessionStatus) Server sends no response (NRC of 0x7F (noActiveSession) is suppressed as described in **Error! Reference source not found.**).
Session A	Session A active	Client sends validly formatted openSession request Server responds with positive response and continues normally. Server shall adapt to any updated parameters in the openSession request (e.g., sessionTimeout, Tx_STmin).
Session A	Session A active	Client sends any request (other than openSession) Request is processed by server and responded to accordingly.
Session A	Session B active	Client sends any request (including openSession) Mismatch detected due to wrong sessionSerialNumber and request is rejected with NRC 0x7D (sessionMismatch). Server closes active session.

7 Core OVTP Function ID Definitions

7.1 REQ-314747/A- openSession (0x01) Function

7.2 REQ-314748/A-Function Description

The openSession request is used to open an OVTP application session to allow application specific functions IDs to be processed. The openSession request defines additional information including the timeout of the session and additional timing information as described below.

7.3 REQ-314749/A-OVTP Header Information

The openSession function shall always have the OVTP header fields set to the following values.

Message	Cntr	CryptoType	SSN
Request	OVTP Application Specific	OVTP Application Specific	1
Response	OVTP Application Specific	OVTP Application Specific	1

When opening a new OVTP session, the client shall generate a 2 byte random number to use as the Session Serial Number in the openSession request and all other requests that occur while that session is open. A server shall store the received value of the Session Serial Number of the openSession request for which it has positively responded to and reject any new request using NRC sessionMismatch where the Session Serial Number in the request does not match. If a client has a need to utilize functional addressing in function IDs which require a Session Serial Number, then the client must use the same Session Serial Number in all of the openSession requests for the ECUs.

7.4 REQ-314750/A-Request Message

7.5 REQ-314751/A-Request message definition

Table 21 — Request message definition

A_Data byte	Parameter Name	Cvt	Byte Value	Mnemonic
#1	openSession Request FID	M	0x01	OS
#2	sessionTimeout	M	0x00 – 0xFF	ST
#3 - #4	Tx_STmin	M	0x0000 – 0xFFFF	TXSTM

7.6 REQ-314752/A-Request message data parameter definition

Table 22 — Request parameter definition

Definition
sessionTimeout

Definition
The sessionTimeout parameter shall indicate the time for which the server shall set its session timeout (e.g., S3$_{server}$). 0x00 = no timeout, session ends at ECU power down or ECU sleep 0x01 – 0xEF : timeout in seconds (1s – 239s) 0xF0 – 0xFE: Reserved for future use 0xFF = no timeout, session persistent until closed (e.g., across key cycles)
Tx_STmin For CAN and CAN-FD: The Tx_STmin parameter allows a client to limit the transmission rate of a server by setting the Tx_STmin (see section **Error! Reference source not found.**). A value of 0x0000 can be used to allow the server to transmit as fast as possible based upon received flow control STmin and ECU configuration settings. OVTP applications may allow a server to limit the range of acceptable Tx_STmin values (e.g., only values of 0ms to 127ms are allowed in an openSession request). If a non-zero value of Tx_STmin is accepted by the server, then this value shall also apply to any publish / subscribe messages which are enabled by OVTP application session (e.g., PARSED Push messages shall utilize the accepted Tx_STmin value in the PARSED openSession request). For Ethernet: This value is a don't care but is recommended to always be set to 0x0000.

7.7 REQ-314753/A-Positive Response Message

7.8 REQ-314754/A-Positive response message definition

Table 23 — Response message definition

A_Data byte	Parameter Name	Cvt	Byte Value	Mnemonic
#1	openSession Response FID	M	0x81	OSPR

7.9 REQ-314755/A-Positive response message parameter definition

No additional positive response message parameters defined.

7.10 REQ-314756/A-Supported negative response codes

The following negative response codes shall be implemented for this function ID. The circumstances under which each response code would occur are documented in Table 24. The listed negative responses shall be used if the error scenario applies to the server.

Table 24 — Supported negative response codes

HEX	Description	Mnemonic
13	**incorrectMessageLengthOrInvalidFormat** The length of the message is wrong.	IMLOIF
22	**conditionsNotCorrect** This code shall be returned if the criteria for the openSession request is not met.	CNC
	requestOutOfRange	ROOR

HEX	Description	Mnemonic
31	The server shall use this response code if it receives a request with an unsupported sessionTimeout or Tx_STmin.	

7.11 REQ-314757/A-message flow example(s) openSession Function

7.12 Example #1 - openSession request / response

Table 25 — request message flow example #1

Message direction	client → server		
Message Type	Request		
A_Data byte	Description (all values are in hexadecimal)	Byte Value	Mnemonic
#1	openSession Request FID	0x01	OS
#2	sessionTimeout = 30s	0x1E	ST
#3 - #4	Tx_STmin = 20ms	0x0014	TXSTM

Table 26 — positive response message flow example #1

Message direction	server → client		
Message Type	Response		
A_Data byte	Description (all values are in hexadecimal)	Byte Value	Mnemonic
#1	openSession Response FID	0x81	OSPR

7.13 REQ-314760/A-closeSession (0x02) Function

7.14 REQ-314761/A-Function Description

The closeSession function is used by the client to close an active session (as opposed to letting a session timeout). It allows for the server to return necessary internal resources and to inform the client that it has successfully returned to a closed session state.

If a closeSession request is received by the server when a session is not open, the server shall respond with a negative response of noActiveSession. If a closeSession request is received by the server when a session is open, but the OVTP embedded Session Serial Number does not match the active Session Serial Number, the server shall respond with a negative response of sessionMismatch and close the active session.

7.15 REQ-314762/A-OVTP Header Information

The closeSession Function ID shall always have the OVTP header fields set to the following values.

Message	Cntr	CryptoType	SSN
Request	OVTP Application Specific	OVTP Application Specific	1
Response	OVTP Application Specific	OVTP Application Specific	1

7.16 REQ-314763/A-Request Message

7.17 REQ-314764/A-Request message definition

Table 27 — Request message definition

A_Data byte	Parameter Name	Cvt	Byte Value	Mnemonic
#1	closeSession Request FID	M	0x02	CS

7.18 REQ-314765/A-Request message parameter definition

No additional request message parameters defined.

7.19 REQ-314766/A-Positive Response Message

7.20 REQ-314767/A-Positive response message definition

Table 28 — Response message definition

A_Data byte	Parameter Name	Cvt	Byte Value	Mnemonic
#1	closeSession Response FID	M	0x82	CSPR

7.21 REQ-314768/A-Positive response message parameter definition

No additional positive response message parameters defined.

7.22 REQ-314769/A-Supported negative response codes

The following negative response codes shall be implemented for this function ID. The circumstances under which each response code would occur are documented in Table 29. The listed negative responses shall be used if the error scenario applies to the server.

Table 29 — Supported negative response codes

HEX	Description	Mnemonic
13	**incorrectMessageLengthOrInvalidFormat**	IMLOIF
	The length of the message is wrong.	
22	**conditionsNotCorrect** This code shall be returned if the criteria for the closeSession request is not met.	CNC
7F	**noActiveSession** The server shall use this response code if there is not a current OVTP application session.	NAS

7.23 REQ-314770/A-Message flow example(s) closeSession Function

7.24 Example #1 - closeSession request / response

Table 30 — request message flow example #1

Message direction	client → server		
Message Type	Request		
A_Data byte	Description (all values are in hexadecimal)	Byte Value	Mnemonic
#1	closeSession Request FID	0x02	CS

Table 31 — positive response message flow example #1

Message direction	server → client		
Message Type	Response		
A_Data byte	Description (all values are in hexadecimal)	Byte Value	Mnemonic
#1	closeSession Response FID	0x82	CSPR

7.25 REQ-314772/A-requestSessionStatus (0x03) Function

7.26 REQ-314773/A-Function Description

The requestSessionStatus function may be used by the client to determine the current server OVTP session status and actively maintain an open OVTP session with a server that has a timed session initiated.
A typical use case of requestSessionStatus is to send periodically (and asynchronously from physical requests) using functional addressing in order to prevent all ECUs on the vehicle from timing out of a timed OVTP application session (see section **Error! Reference source not found.** for more details).

7.27 REQ-314774/A-OVTP Header Information

The requestSessionStatus function shall always have the OVTP header fields set to the following values.

Message	**Cntr**	**CryptoType**	**SSN**
Request	0	000	0
Response	0	000	0

7.28 REQ-314775/A-Request Message

7.29 REQ-314776/A-Request message definition

Table 32 — Request message definition

A_Data byte	Parameter Name	Cvt	Byte Value	Mnemonic
#1	requestSessionStatus Request FID	M	0x03	RSS
#2	suppressResponseIndication	M	0x00 – 0x80	SRI

7.30 REQ-314777/A-Request message parameter definition

Table 33 — Request parameter definition

Definition
suppressResponseIndication The suppressResponseIndication parameter is a state encoded parameter with the following states defined: 0x00 = Send all responses (i.e., send positive or negative response) 0x80 = Suppress positive response (i.e., send response only if negative)

7.31 REQ-314778/A-Positive Response Message

7.32 Positive response message definition

Table 34 — Response message definition

A_Data byte	Parameter Name	Cvt	Byte Value	Mnemonic
#1	requestSessionStatus Response F ID	M	0x83	RSSPR
#2	sessionStatus	M	0x01 – 0x02	SS
 #3 #4	sessionSerialNumber [] = byte#1 (MSB) byte#2 (LSB)]	C	 0x00 – 0xFF 0x00 – 0xFF	SSN B1 B2
C: This parameter is mandatory to be present if a session is open (i.e., sessionStatus = 0x01). This parameter shall not be present if a session is not currently open (i.e., sessionStatus = 0x02).				

7.33 REQ-314781/A-Positive response message parameter definition

Table 35 — Response message data-parameter definition

Definition
sessionStatus The sessionStatus parameter is a state encoded value which indicates whether or not an OVTP application session is currently active. 0x01 = A session is currently active for the OVTP application 0x02 = No session is active for the OVTP application
sessionSerialNumber The sessionSerialNumber parameter is the value of the sessionSerialNumber associated with the currently active OVTP application session.

7.34 REQ-314782/A-Supported negative response codes

The following negative response codes shall be implemented for this function. The circumstances under which each response code would occur are documented in Table 36. The listed negative responses shall be used if the error scenario applies to the server.

Table 36 — Supported negative response codes

NRC	Description	Mnemonic
13	**incorrectMessageLengthOrInvalidFormat** The length of the message is wrong.	IMLOIF
31	**requestOutOfRange** The server shall use this response code if it receives a request with an unsupported suppressResponseIndication value (i.e., not 0x00 or 0x80).	ROOR

7.35 REQ-314783/A-Message flow example(s) requestSessionStatus Function

7.36 Example #1 - requestSessionStatus request / response with active session

Example #1 assumes the ECU has an open OVTP session for the given application with sessionSerialNumber equal to 0xABCD.

Table 37 — request message flow example #1

Message direction	client → server		
Message Type	Request		
A_Data byte	Description (all values are in hexadecimal)	Byte Value	Mnemonic
#1	requestSessionStatus Function ID	0x03	RSS
#2	suppressResponseIndication	0x00	SRI

Table 38 — positive response message flow example #1

Message direction	server → client		
Message Type	Response		
A_Data byte	Description (all values are in hexadecimal)	Byte Value	Mnemonic
#1	requestSessionStatus Response Function ID	0x83	RSS
#2	sessionStatus	0x01	SS
#3 - #4	sessionSerialNumber	0xABCD	SSN

7.37 Example #2 - requestSessionStatus request / response with no active session

Example #2 assumes the ECU does not have an OVTP session for the given application.

Table 39 — request message flow example #1

Message direction	client → server		
Message Type	Request		
A_Data byte	Description (all values are in hexadecimal)	Byte Value	Mnemonic
#1	requestSessionStatus Function ID	0x03	RSS
#2	suppressResponseIndication	0x00	SRI

Table 40 — positive response message flow example #1

Message direction	server → client		
Message Type	Response		
A_Data byte	Description (all values are in hexadecimal)	Byte Value	Mnemonic
#1	requestSessionStatus Response Function ID	0x83	RSS
#2	sessionStatus	0x02	SS

7.38 REQ-314786/A-Negative Response (0x7F) Function

7.39 REQ-314787/A-Function Description

The Negative Response function provides the generic structure that all OVTP functions shall use for reporting back errors to requests. The function shall report the failed function ID, a negative response code (NRC), and any associated metadata that can describe more details of the failure. All negative response codes are defined in Table 43. Any additional NRC metadata for a given NRC on a given OVTP application shall be defined by the OVTP application specific implementation document and may vary per function ID.

7.40 REQ-314788/A-OVTP Header Information

The negative response shall always have the OVTP header fields set to the following values.

Message	**Cntr**	**CryptoType**	**SSN**
Response	Request Dependent	Request Dependent	Request Dependent

The OVTP header information fields in the negative response shall always have the same values as the OVTP request message with elicited the negative response. For example, if the request message used a Cntr of 0, a CryptoType of 0, and an SSN of 1, then the response shall use a Cntr of 0, a CryptoType of 0, and an SSN of 1. If the request message used an SSN of 1 and no active session is open in the server, the corresponding negative response shall include the two byte session serial number that was included in the request.

7.41 REQ-314789/A-Generic Negative Response Message Format

7.42 REQ-314790/A-Negative response message definition

Table 41 — Response message definition

A_Data byte	Parameter Name	Cvt	Byte Value	Mnemonic
#1	Negative response FID	M	0x7F	NR
#2	failedFID	M	0x00 – 0x7E	FF
#3	negativeResponseCode	M	0x00 – 0xFF	NRC
 #4 : #n	NRCMetaData [] = NRCMetaData#1 : NRCMetaData#p	 O : O	 0x00 – 0xFF : 0x00 – 0xFF	NMD

7.43 REQ-314791/A-Message parameter definition

Table 42 — Response message data-parameter definition

Definition
failedFID

Definition
The failedFID parameter indicates which request Function ID the negative response is rejecting.
negativeResponseCode The negativeResponseCode parameter provides details on the high level reason that a function was not accepted. See Table 43 for allowed values.
NRCMetaData The NRCMetaData parameter allows an OVTP application to provide additional details (beyond the NRC) to assist in troubleshooting why a specific function request was rejected. This information is OVTP application specific and, if required, shall be captured by the specific OVTP application implementation document. An example use case for this could be to return a percent complete value when using NRC 0x78 (requestCorrectlyReceived-ResponsePending).

Table 43 defines all negative response codes used within this standard. Each function ID specifies the applicable negative response codes. Although NRCMetaData may be specific to a given function ID and a given application, it is the intent of this standard that NRCs numbers themselves shall have consistent meaning regardless of the OVTP application. Therefore, any OVTP application needs for new NRCs must be coordinating with this specification to avoid conflict and for future incorporation.

Table 43 — Negative Response Code (NRC) definition and values

Byte value	Negative Response Code (NRC) definition	Mnemonic
0x00	**positiveResponse** This NRC shall not be used in a negative response message. This positiveResponse parameter value is reserved for server internal implementation.	PR
0x01 – 0x0F	**reserved** This range of values is reserved by this document for future definition.	RSV
0x10	**generalReject** This NRC indicates that the requested action has been rejected by the server. The generalReject response code shall only be implemented in the server if none of the negative response codes defined in this document meet the needs of the implementation. At no means shall this NRC be a general replacement for the response codes defined in this document.	GR
0x11	**functionNotSupported** This NRC indicates that the requested action will not be taken because the server does not support the requested function. The server shall send this NRC in case the client has sent a request message with a function identifier which is unknown, not supported by the server, or is specified as a response function identifier. Therefore this negative response code is not shown in the list of negative response codes to be supported for a function, because this negative response code is not applicable for supported functions. IMPORTANT: This NRC shall be suppressed (i.e., no response sent) if the request was sent with functional addressing.	FNS
0x13	**incorrectMessageLengthOrInvalidFormat** This NRC indicates that the requested action will not be taken because the length of the received request message does not match the prescribed length for the specified function or the format of the parameters do not match the prescribed format for the specified function.	IMLOIF
0x14	**responseTooLong** This NRC shall be reported by the server if the response to be generated exceeds the maximum number of bytes available by the underlying network layer. This could occur if the response message exceeds the maximum size allowed by the underlying transport protocol or if the response message exceeds the server buffer size allocated for that purpose. While not common, this NRC may be supported by any function ID where the request has a variable number of parameters that affects the response size and therefore is not listed in the list of applicable response codes of each function.	RTL
0x15	**endToEndSignatureInvalid** This NRC indicates that an embedded end to end signature in the function was deemed invalid by the server.	ETESI
0x16	**ESNInvalid** This NRC indicates that an embedded electronic serial number in the function was deemed invalid by the server.	ESNI
0x17	**softwareUpdateCounterInvalid** This NRC indicates that an embedded software update counter in the function was deemed invalid by the server.	SUCI
0x18 – 0x20	**reserved** This range of values is reserved by this document for future definition.	RSV
0x21	**busyRepeatRequest** This NRC indicates that the server is temporarily too busy to perform the requested operation. In this circumstance the client shall perform repetition of the "identical request message" or "another request message". The repetition of the request shall be delayed by a time specified in the respective implementation documents. This NRC is in general supported by each function ID, as not otherwise stated in the OVTP application implementation document, therefore it is not listed in the list of applicable response codes of each function.	BRR

Byte value	Negative Response Code (NRC) definition	Mnemonic
0x22	**conditionsNotCorrect** This NRC indicates that the requested action will not be taken because the server prerequisite conditions are not met.	CNC
0x23	**reserved** This range of values is reserved by this document for future definition.	RSV
0x24	**requestSequenceError** This NRC indicates that the requested action will not be taken because the server expects a different sequence of request messages as sent by the client. This may occur when sequence sensitive requests are issued in the wrong order.	RSE
0x25 – 0x30	**reserved** This range of values is reserved by this document for future definition.	RSV
0x31	**requestOutOfRange** This NRC indicates that the requested action will not be taken because the server has detected that the request message contains a parameter which attempts to substitute a value beyond its range of authority (e.g. attempting to substitute a data byte of 111 when the data is only defined to 100), or which attempts to access a request parameter that is not supported.	ROOR
0x32	**reserved** This range of values is reserved by this document for future definition.	RSV
0x33	**securityRequired** This NRC indicates that the requested action will not be taken because the received request requires a security strategy which has not yet been satisfied by the client.	SR
0x34 – 0x6F	**reserved** This range of values is reserved by this document for future definition.	RSV
0x70	**downloadNotAccepted** This NRC indicates than an attempt to download to a server’s memory cannot be accomplished due to some fault conditions.	DNA
0x71	**transferDataSuspended** This NRC indicates that a data transfer operation was halted due to some fault.	TDS
0x72	**generalProgrammingFailure** This NRC indicates that the server detected an error when erasing or programming a memory location.	GPF
0x73	**wrongSequenceCounter** This NRC indicates that the server detected an error in an embedded sequence counter that is part of the function ID definition.	RSV
0x74 – 0x77	**reserved** This range of values is reserved by this document for future definition.	RSV
0x78	**requestCorrectlyReceived-ResponsePending** This NRC indicates that the request message was received correctly, and that all parameters in the request message were valid, but the action to be performed is not yet completed and the server is not yet ready to receive another request. As soon as the requested function ID has been completed, the server shall send a positive response message or negative response message with a response code different from this. The negative response message with this NRC may be repeated by the server until the requested function ID is completed and the final response message is sent. This NRC might impact the application layer timing parameter values. This NRC shall only be used in a negative response message if the server will not be able to receive further request messages from the client while completing the requested function. When this NRC is used, the server shall always send a final response (positive or negative). This NRC is in general supported by each function ID, as not otherwise stated in the OVTP application implementation document, therefore it is not listed in the list of applicable response codes of each function.	RCRRP
0x79	**validationFailed** This NRC indicates that the validation check contained in the request was executed and returned a result indicating that the validation check failed.	RSV
0x7A – 0x7C	**reserved** This range of values is reserved by this document for future definition.	RSV

Byte value	Negative Response Code (NRC) definition	Mnemonic
0x7D	**sessionMismatch** This NRC indicates that the sessionSerialNumber included in the OVTP header does not match the expected sessionSerialNumber for the ECU's currently active session. This NRC is in general supported by each function ID, as not otherwise stated in the OVTP application implementation document, therefore it is not listed in the list of applicable response codes of each function.	SM
0x7E	**reserved** This range of values is reserved by this document for future definition.	RSV
0x7F	**noActiveSession** This NRC indicates that the requested action will not be taken because the server requires an active session to support the requested function and no session is active. This NRC shall only be used when the requested function ID is known to be supported by the ECU. This NRC is in general supported by each function ID, as not otherwise stated in the OVTP application implementation document, therefore it is not listed in the list of applicable response codes of each function. IMPORTANT: This NRC shall be suppressed (i.e., no response sent) if the request was sent with functional addressing.	NAS
0x80 – 0xFF	**reserved** This range of values is reserved by this document for future definition.	RSV