Function Specification (FncS)

()

Document Type	Function Specification (FncS)	
Document ID	547919	
Document Location	VSEM Rich Client, VSEM Active Workspace	
Document Owner	Nasser, Mohamad (mnasse11)	
Document Version	I	
Document Status		
Date Issued	23-Nov-2020 19:15	
Date Revised		
Document Classification	GIS1 Item Number:	
	GIS2 Classification:	

Document Approval			
Person	Role	Email Confirmation	Date

Content

List of Figures

List of Tables

1 Document Overview

Ford's connectivity vision includes the ability to provide over the air 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 in-vehicle client to multiple ECUs needs to be developed to provide a framework for efficiently and securely transmitting the data within the vehicle. To support this goal, Ford has designed a protocol called On-Vehicle Telematics Protocol (OVTP) as described in reference [1], which supports various applications including Over the Air updates. This document builds upon the OVTP specification and describes the functional use case of OTA over OVTP and is the controlling document for OTA Function IDs.

1.1 Release History

Date	Version	Notes
2017-05-11	003	Official Spec Release
2017-09-13	003.1	Draft Release
2017-09-14	004	Official Spec Release
2017-10-11	004.1	Draft Release
2018-02-08	004.4	Draft Release
2018-03-16	005	Official Spec Release
2018-05-17	005.1	Draft Release
2018-06-11	005.2	Draft Release
2018-08-24	005.3	Draft Release
2018-08-30	005.4	Draft Release
2018-11-16	006	Official Spec Release to incorporate clarifications and editorial changes
2019-08-16	006.1	Draft Release
2019-10-01	006.2	Draft Release
2019-10-08	006.3	New draft with changes up to 2019-10-08 (DID $D03F support)
2019-11-14	006.4	New draft with F4 timing parameter changes
2020-03-04	007	Official Spec Release to incorporate F2/F4 timing updates (REQ-333416, REQ-333418) Added DID D039 to the spec (REQ-383103)
2020-03-24	007.1	Draft release. Update: "Clear DTC" is removed from sample sequence diagram in initiateActivation(0x1C) and initiateRollback(0x1D) functions.
2020-07-22	007.2	Changes: 1. FID 0x1F - removed from spec. 2. NRC 0x11 (functionNotSupported) - removed from all FIDs, except for Diffupdate FID (0x18) 3. For FID 0x1D, NRC 0x72 is added. 4. NRC 0xF8 InCompatibleSoftware – added for 0x1A, 0x1B and 0x1C FIDs 5. REQ-349071 is updated for general NRCs (0x10, 0x21)
2020-09-24	008	REQ-308098 updated for clarification. Official release of 007.2 as 008
2020-11-23	008.1	Added Notes to Appendix D related to SWASH Calculation

1.2 Scope

The On Vehicle Telematics Protocol OTA Function Definition Document defines the vehicle system requirements that each vehicle module must comply with in order to utilize the function IDs that are defined in this document for the purpose of providing over the air software updates.

This document does not apply to normal vehicle signaling on the vehicle's communication data link between two Electronic Control Units, or standard diagnostic message transmission between a diagnostic tester and an ECU.

1.3 Acronyms

Table 1 — Acronyms Definition

CAN	Controller Area Network
ECG	Enhanced Central Gateway
ECU	Electronic Control Unit
FNOS	Ford Network Operating System
FTCP	Ford Telematics Communication Protocol
M	Mandatory
MC	Mandatory on Condition
OTA	Over The Air
OVTP	On Vehicle Telematics Protocol
TCU	Telematics Control Unit
WERS	Worldwide Engineering Release System

1.4 Timing Parameters

REQ-333416/C-OVTP Timing Parameters

The OTA OVTP implementations shall set up the following timing parameters to be used in the OVTP communication protocol. Refer to reference [1] for definitions of these parameters.

Table 2 — OVTP Timing Parameters

Timing Parameter	Minimum	Maximum
ΔF2	0 ms	100ms
$F2_{Server}$	0 ms	350ms
$F2^*_{Server}$	0 ms	10000ms
$F4_{Server}$	$F2_{Server}$	OVTP Function ID Dependent

REQ-333417/C-OTA STmin Requirements

The STmin parameter in flow control frames transmitted by an ECU is specified by the ECU owner, but the parameter shall never be greater than 2 milliseconds on Classical CAN networks and shall never be greater than 3 milliseconds on CAN FD networks, unless there is explicit approval from FORD CV&S.

REQ-333418/E-F4 Requirements

Table 3 — Requirements on maximum value of F4

OVTP FUNCTION ID(Note 4)		$F4_{Server\_Max}$	Notes
Name	HEX		
openSession	0x01	$F2_{Server\_Max}$	
closeSession	0x02	750ms	

requestSessionStatus	0x03	$F2_{Server\_Max}$	
readOTADataByIdentifier	0x11	$F2_{Server\_Max}$	
authorizeEraseMemory	0x12	2000ms	
eraseMemory	0x13	120000ms	See Note 1
authorizeDownload	0x14	2000ms	
initiateDownload	0x15	10000ms	
transferData	0x16	10000ms	
completeDownload	0x17	1000ms	
diffUpdate	0x18	120000ms	See Note 2
validateLogicalBlock	0x19	5000ms	See Note 3
prepareActivation	0x1A	120000ms	See Note 1
authorizeActivation	0x1B	5000ms	
initiateActivation	0x1C	$F2_{Server\_Max}$	
initiateRollBack	0x1D	5000ms	
initiateForceSyncCounter	0x1E	2000ms	
performAuthorizedActivity	0x1F	5000ms	

Notes:

1 eraseMemory (FID 0x13) and prepareActivation (FID 0x1A) may increase **$F4_{Server\_Max}$** time 120s for each megabyte range of data over 1 megabyte that is processed by the request. For example, an eraseMemory request with a memorySize value between 1 and 2 megabytes may utilize a **$F4_{Server\_Max}$** time of 240s and an eraseMemory request with a memorySize value between 2 and 3 megabytes may utilize **$F4_{Server\_Max}$** time of 360s.

2 diffUpdate (FID 0x18) may increase **$F4_{Server\_Max}$** time 120s for each megabyte range of data over 1 megabyte that is being updated (Process, Erase and Program into the inactive memory).

3 validateLogicalBlock (FID 0x19) To account for checksum calculation times, a given download block may increase **$F4_{Server\_Max}$** time 5s for each 50K of data over 50K that is sent in a single block. For example, validating a logical block under 50K would allow 5s, validating a logical block between 50K and 100K would allow 10s, and validating a logical block between 100K and 150k would allow 15s, etc.

4 Note that these requirements are applicable only to services that are supported by the ECU. Unsupported services shall always utilize a $F4_{Server\_max}$ value equal to $F2_{Server\_Max}$ (i.e., NRC 78_H not allowed). Exceptions to $F4_{Server\_max}$ are allowed but only when approved by Ford Core OTA.

5 Note that F4 timings are performance requirements. Exceptions to this are possible but require review and explicit approval of the details of the design by the core IVSU OTA team.

2 Over the Air Application

REQ-333392/B-Overview

Table 4 — OVTP Core Functions for OTA

Function ID	Function Name	Description	Cvt
0x01	openSession	See ref [1]	M
0x02	closeSession	See ref [1]	M
0x03	requestSessionStatus	See ref [1]	M

Table 5 — OTA Application Functions

Function ID	Function Name	Description	Cvt
0x11	readOTADataByIdentifier	The readOTADataByIdentifier function provides the client with the ability to read server specific information such as part numbers or internal software state information (e.g., current progress of an OTA download via DID 0xD022).	M
0x12	authorizeEraseMemory	The authorizeEraseMemory function provides the client the interface for authorizing the erase of memory (FID 0x13).	MC1
0x13	eraseMemory	The eraseMemory function provides the client the interface for erasing the module's memory.	MC1
0x14	authorizeDownload	The authorizeDownload function provides the client the interface for authorize the programming of the module (FID 0x15). This function will be signed by the cloud to provide proper security authorizations for Over the Air updates.	MC1
0x15	initiateDownload	The initiateDownload function provides the client the interface for initiating the download of data to the target ECU.	MC1
0x16	transferData	The transferData function provides the client the ability to transmit a block of data to the specific module for storage in memory.	MC1
0x17	completeDownload	The completeDownload function provides the client the ability to inform the specific module that it has completed transferring all the data to the module.	MC1
0x18	diffUpdate	The diffUpdate function provides the client the ability to request the server to unpack and process a differential file.	MC2
0x19	validateLogicalBlock	The validateLogicalBlock function provides the client the ability to request the server to perform the signature check on the software verification structure and allow the client to confirm that the server has verified the authenticity of this signed portion of software prior to activating.	MC1
0x1A	prepareActivation	The prepareActivation function provides the client the interface for ensuring everything is prepared for an upcoming activation.	MC1
0x1B	authorizeActivation	The authorizeActivation function provides the client the interface for authenticating a swap activation operation to occur.	MC1
0x1C	initiateActivation	The initiateActivation function provides the client the interface for initiating a swap activation operation to occur on a target module(s).	MC1
0x1D	initiateRollBack	The initiateRollBack function provides the client the interface to force a memory rollback of a target ECU.	MC1
0x1E	initiateForceSyncCounter	The initiateForceSyncCounter function provides the client the interface to force an ECU to increment its OTA specific software update counter. This is used to invalidate certain signed functions that are no longer needed (e.g., to prevent replay).	MC1
0x1F	performAuthorizedActivity	The performAutorizedActivity function provides the client the interface to request the OTA server perform a specific function/activity.	MC3

MC1 only applicable to server with OTA capability.
MC2 only applicable to server with diff update capability.
MC3 only applicable to servers explicitly required to implement one or more activityIdentifiers defined by FID 0x1F.

2.1 OTA Program Flow

The following figure shows a high level overview of a typical example OTA programming flow. Refer to the individual figures for more detail on each portion of the OTA process.

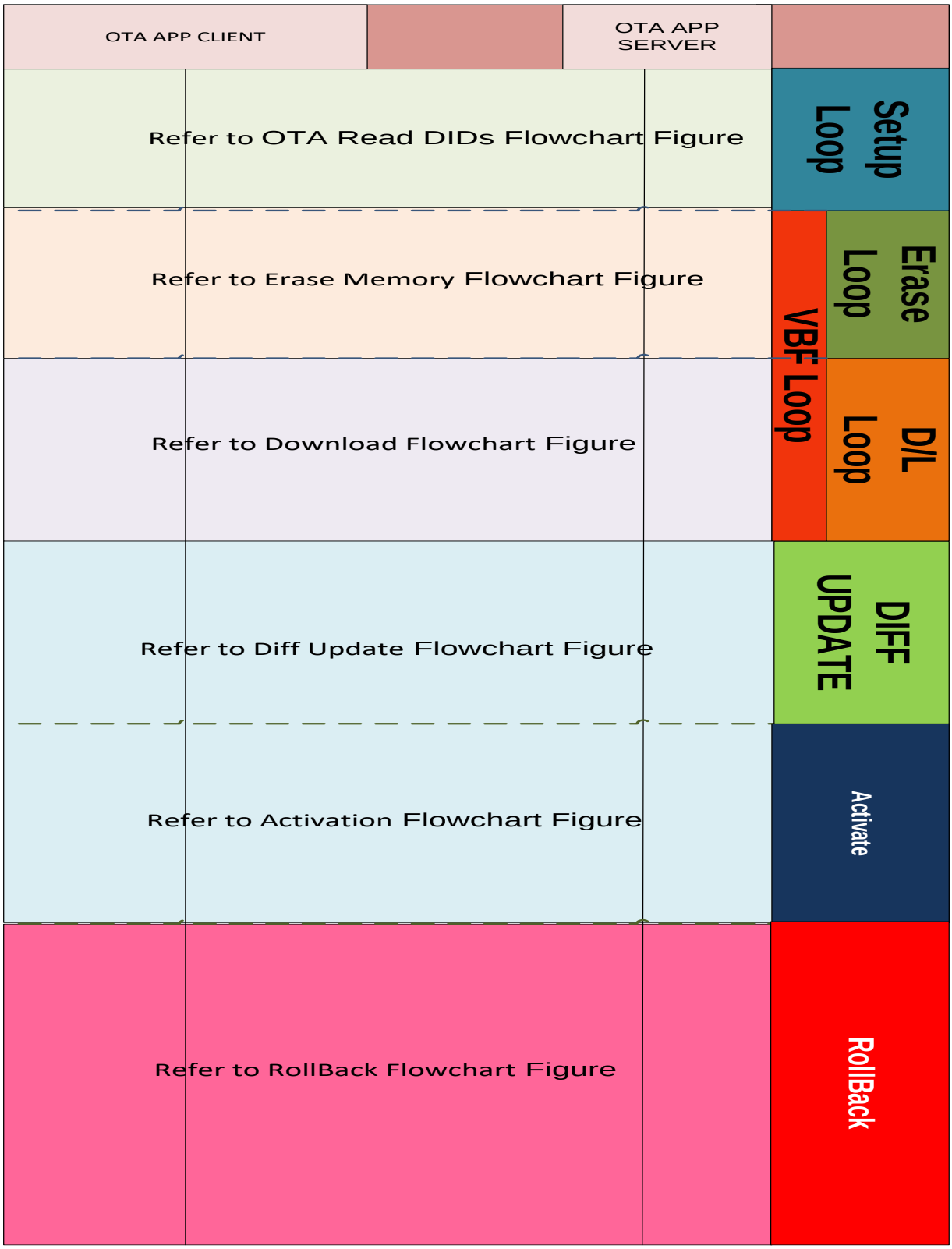


Figure 1: Sequence Diagram for OTA OVTP Functions

REQ-333407/C-Signed OTA Functions

Certain OTA functions contain a signature within the data. The details of how the signature is calculated and evaluated are described within reference [2]. Unless otherwise specified in this document, all signatures shall be calculated using RSASSA-PSS and all hashes shall be calculated using SHA-256. The data to be signed shall always include all data in the request message starting at the Function ID byte to the byte immediately preceding the signature. When a signed OTA function ID is received within an OVTP session, a positive response shall only occur when the included SUcounter is greater than the value received in the last initiateForceSyncCounter that the ECU has positively responded to (or greater than the default value of the SUcounter if initiateForceSyncCounter has never been received), the FESN in the request matches the expected FESN in the server, and the signature validates successfully as described in reference [2].

There are two categories of these signed OTA functions. Category 1 is where the desired end action which is protected by the signature is performed in the same request that the signature is within. Examples of this include prepareActivation and initiateRollback. Category 2 is where the desired end action which is protected by the signature is only authorized in the request that the signature is within, and therefore a separate request must be received to perform this authorized action. For category 2, the authorization request and the authorized action request come in pairs. Currently, there are 3 instances of this category of signed OTA functions within the spec. These are authorizeEraseMemory paired with eraseMemory, authorizeDownload paired with InitiateDownload, and authorizeActivation paired with initiateActivation.

This section generically addresses the required period of validity of the corresponding authorized action when a validly formatted “authorizeFunctionX” is positively responded to by the server for category 2 signed OTA functions described above. In order to best describe this generically, this section refers to the signed OTA function as authorizeFunctionX, and the corresponding authorized action as initiateFunctionX. Once an authorizeFunctionX has been positively responded to, the corresponding initiateFunctionX shall be authorized until the current OVTP session ends or until a new signed OTA function ID is received by the OTA application. When either of these events occur, the previously authorized initiateFunctionX shall no longer be authorized and additionally shall be considered not active if previously sent (e.g., initiateDownload always deactivated). Reception of a valid openSession request with the same sessionSerialNumber of an active OVTP session shall be considered a continuation of the current OVTP session, and therefore not affect any existing authorization.

REQ-349071/C-OTA Function IDs

General requirements that may apply to all OTA over OVTP functions are specified in this section. The following sections describe the request and response structure and high level requirements for all OTA over OVTP function IDs.

OTA over OVTP messages can be transmitted using physical or functional addressing, so the sever shall support reception of both physically and functionally addressed requests.

0x10 - GeneralReject and 0x21 - busyRepeatRequest NRCs are specified in Core OVTP specification. These NRCs shall be supported by all FIDs specified in this specification.

REQ-308095/C-openSession (0x01) Function

.

2.1.1.1 Function Description

Refer to reference [2] for details on this function.

2.1.1.2 OVTP Header Information

For OTA over OVTP, the openSession function shall always have the OVTP header fields set to the following values.

Table 6 — openSession Header Info

Message	Cntr	CryptoType	SSN
Request	0	0	1
Response	0	0	1

2.1.1.3 Request Information

ECU must accept sessionTimeout values in the range of 0x00 – 0xEF. sessionTimeout values of 0xF0 – 0xFF shall never be supported.

REQ-308096/C-closeSession (0x02) Function

2.1.1.4 Function Description

Refer to reference [2] for details on this function.

2.1.1.5 OVTP Header Information

For OTA over OVTP, the closeSession function shall always have the OVTP header fields set to the following values.

Table 7 — closeSession Header Info

Message	Cntr	CryptoType	SSN
Request	0	0	1
Response	0	0	1

REQ-308097/B-requestSessionStatus (0x03) Function

2.1.1.6 Function Description

Refer to the reference [2] for details on this function.

2.1.1.7 OVTP Header Information

For OTA over OVTP, the requestSessionStatus function shall always have the OVTP header fields set to the following

Table 8 — requestSessionStatus Header Info

Message	Cntr	CryptoType	SSN
Request	0	0	0
Response	0	0	0

REQ-333394/A-Negative Response (0x7F) Function

2.1.1.8 Function Description

Refer to reference [2] for details on this function.

If an ECU is actively processing an OTA over OVTP function ID that supports NRC 0x20 (requestProcessingSuspended-RepeatRequest) and determines that the network is ready to begin the sleep process, the ECU shall suspend the action associated with that OTA function ID and respond with NRC 0x20 within 500ms.

REQ-308098/H-readOTADataByIdentifier (0x11) Function

2.1.1.9 Function Description

The readOTADataByIdentifier function is used to read out information deemed relevant for performing an OTA update of an ECU. For example, it may be used for confirmation that a server is at a given level of hardware or is executing a given level of software. This function leverages existing supported diagnostic dataIdentifiers (DIDs) as the mechanism to read information for OTA. The server shall specify which DIDs are supported via this function. The following table outlines the mandatory DIDs this function shall support. Additional DIDs shall be included on a per module basis for OTA as needed for the individual ECU. Any implemented DIDs that report WERS released software part numbers (e.g., ECU Software #6 Part Number) which are defined after the publication of this specification shall be reportable via this function if supported by the ECU. DIDs which require diagnostic securityAccess to read using diagnostic services (e.g., service 0x22 readDataByIdentifier) shall not be reportable using this function.

The server may limit the number of dataIdentifiers that can be simultaneously requested. The server shall support at least two dataIdentifiers requested in a single request and transmit their value in one single positive response containing the associated dataRecord parameter(s). The request message may contain the same dataIdentifier multiple times. The server shall treat each dataIdentifier as a separate parameter and respond with data for each dataIdentifier as often as requested.

In the case when multiple dataIdentifiers are requested in a single request, a single supported DID of those requested is sufficient to generate a positive response. The response only includes the dataIdentifiers that are supported. A negative response will be generated if all of the dataIdentifiers in the request are not supported.

All dataIdentifiers supported by this functionID shall always be readable using diagnostic service readDataByIdentifier (service 22_H). Furthermore, to ensure consistency of data, a given dataIdentifier shall read its data from the same storage location or by using the same common function or calculation routine when reported via this function as is used when reported via diagnostic service readDataByIdentifier.

Table 9 — Required DID Support

DID (Hex)	Name	Cvt
F111	ECU Core Assembly Number	M
F113	ECU Delivery Assembly Number	M
F188	Vehicle Manufacturer ECU Software Number	MC1
F120	ECU Software #2 Part Number	MC1
F121	ECU Software #3 Part Number	MC1
F122	ECU Software #4 Part Number	MC1
F123	ECU Software #5 Part Number	MC1
F124	ECU Main Calibration Data Number	MC1
F125	ECU Calibration Data #2 Number	MC1
F126	ECU Calibration Data #3 Number	MC1
F127	ECU Calibration Data #4 Number	MC1
F128	ECU Calibration Data #5 Number	MC1
F108	ECU Network Signal Calibration Number	MC1
F10A	ECU Cal-Config Part Number	MC1
F16B	ECU Cal-Config #2 Part Number	MC1
F16C	ECU Cal-Config #3 Part Number	MC1
F16D	ECU Cal-Config #4 Part Number	MC1
F16E	ECU Cal-Config #5 Part Number	MC1
F17D	ECU Cal-Config #6 Part Number	MC1
F10E	ECU Cal-Config #7 Part Number	MC1
F17F	Ford ESN	MC2
D022	In Progress OTA Download Address	MC2
D026	OTA Activation Preconditions	MC2
D029	OTA over OVTP Support Level	M
D02B	OTA Software Update Counter	MC2
D031	Vehicle Conditions Preventing OTA Activation or OTA SWDL of New Software	O
D039	OTA Partition Status	MC2
D03B	OTA Debug Information	MC2
D03E	In-Use OTA Command Signing Public Key Hash	MC2
D03F	In-Use Application Signing Public Key Hash	MC2
D04F	OTA Programming Session Entry and A/B Activation Precondition Status	M

MC1: Mandatory if the dataIdentifier is readable by the ECU with diagnostic services.
MC2: Mandatory if the ECU is OTA updateable.

Note: Above DIDs shall be readable via diagnostic services (SID $22) in Default application session.

2.1.1.10 OVTP Header Information

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

Table 10 — readOTADataByIdentifier Header Info

Message	Cntr	CryptoType	SSN
Request	0	0	1
Response	0	0	1

2.1.1.11 Request Message

Table 11 — Request message definition

A_Data byte	Parameter Name	Cvt	Byte Value	Mnemonic
#1	readOTADataByIdentifier FID	M	0x11	ROTAD
	dataIdentifier#1[] = [	M	0x00 – 0xFF	DID_
#2	Byte#1 (MSB)	M	0x00 – 0xFF	HB
#3	Byte#2]			LB
:	:	:	:	:
	dataIdentifier#m[] = [			DID_
#n-1	Byte#1 (MSB)	O	0x00 – 0xFF	HB
#n	Byte#2]	O	0x00 – 0xFF	LB

Table 12 — Request message parameter definition

Definition
dataIdentifier This parameter identifies the diagnostic server data record that is being requested by the client. The dataIdentifier size and definition shall be consistent with the dataIdentifiers supported using diagnostics.

2.1.1.12 Positive Response Message

Table 13 — Positive Response message definition

A_Data byte	Parameter Name	Cvt	Byte Value	Mnemonic
#1	readOTADataByIdentifier Response FID	M	0x91	ROTADPR
	dataIdentifier#1[] = [			DID_
#2	Byte#1 (MSB)	M	0x00 – 0xFF	HB
#3	Byte#2]	M	0x00 – 0xFF	LB
	dataRecord[] = [			DREC_
#4	Byte#1 (MSB)	MC	0x00 – 0xFF	DATA_1
	:	:	:	:
#(k-1)+4	Byte#k]	MC	0x00 – 0xFF	DATA_K
:	:	:	:	:
	dataIdentifier#m[] = [			DID_
#n-(o-1)-2	Byte#1 (MSB)	O	0x00 – 0xFF	HB
#n-(o-1)-1	Byte#2]	O	0x00 – 0xFF	LB
	dataRecord[] = [			DREC_
#n-(o-1)	Byte#1 (MSB)	O	0x00 – 0xFF	DATA_1
	:	:	:	:
#n	Byte#o]	O	0x00 – 0xFF	DATA_K

Table 14 — Positive Response message parameter definition

Definition
dataIdentifier This parameter is an echo of the data parameter dataIdentifier from the request message
dataRecord This parameter is used by the readOTADataByIdentifier positive response message to provide the requested DID information to the client.

2.1.1.13 Supported negative response codes

Table 15 — Supported negative response codes

NRC	Description	Mnemonic
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.	IMLOIF
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).	RTL
0x31	**requestOutOfRange** This NRC shall be sent if none of the requested dataIdentifier values are supported by the device	ROOR

2.1.1.14 Message flow example(s)

Table 16 — readOTADataByIdentifier 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	readOTADataByIdentifier Request FID	0x11	ROTAD
#2	DataIdentifier#1[1]	0xF1	DID
#3	DataIdentifier#1[2]	0x11	DID
#4	DataIdentifier#2[1]	0xF1	DID
#5	DataIdentifier#2[2]	0x88	DID

Table 17 — 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	readOTADataByIdentifier Response FID	0x92	ROTADPR
#2	Data Identifier#1[1]	0xF1	DID
#3	Data Identifier#1[2]	0x11	DID
#4	Data Record[1]	0x33	DREC
#5	Data Record[2]	0x33	DREC
#6	Data Record[3]	0x33	DREC
#7	Data Record[4]	0x33	DREC
#8	Data Record[5]	0x33	DREC
#9	Data Record[6]	0x33	DREC
#10	Data Record[7]	0x33	DREC
#11	Data Record[8]	0x33	DREC
#12	Data Record[9]	0x00	DREC
#13	Data Record[10]	0x00	DREC
#14	Data Record[11]	0x00	DREC
#15	Data Record[12]	0x00	DREC
#16	Data Record[13]	0x00	DREC
#17	Data Record[14]	0x00	DREC

Message direction	server → client		
Message Type	Response		
A_Data byte	Description (all values are in hexadecimal)	Byte Value	Mnemonic
#18	Data Record[15]	0x00	DREC
#19	Data Record[16]	0x00	DREC
#20	Data Record[17]	0x00	DREC
#21	Data Record[18]	0x00	DREC
#22	Data Record[19]	0x00	DREC
#23	Data Record[20]	0x00	DREC
#24	Data Record[21]	0x00	DREC
#25	Data Record[22]	0x00	DREC
#26	Data Record[23]	0x00	DREC
#27	Data Record[24]	0x00	DREC
#28	Data Identifier#2[1]	0xF1	DID
#29	Data Identifier#3[2]	0x88	DID
#30	Data Record[1]	0x34	DREC
#31	Data Record[2]	0x34	DREC
#32	Data Record[3]	0x34	DREC
#33	Data Record[4]	0x34	DREC
#34	Data Record[5]	0x34	DREC
#35	Data Record[6]	0x34	DREC
#36	Data Record[7]	0x34	DREC
#37	Data Record[8]	0x34	DREC
#38	Data Record[9]	0x00	DREC
#39	Data Record[10]	0x00	DREC
#40	Data Record[11]	0x00	DREC
#41	Data Record[12]	0x00	DREC
#42	Data Record[13]	0x00	DREC
#43	Data Record[14]	0x00	DREC
#44	Data Record[15]	0x00	DREC
#45	Data Record[16]	0x00	DREC
#46	Data Record[17]	0x00	DREC
#47	Data Record[18]	0x00	DREC
#48	Data Record[19]	0x00	DREC
#49	Data Record[20]	0x00	DREC
#50	Data Record[21]	0x00	DREC
#51	Data Record[22]	0x00	DREC
#52	Data Record[23]	0x00	DREC
#53	Data Record[24]	0x00	DREC

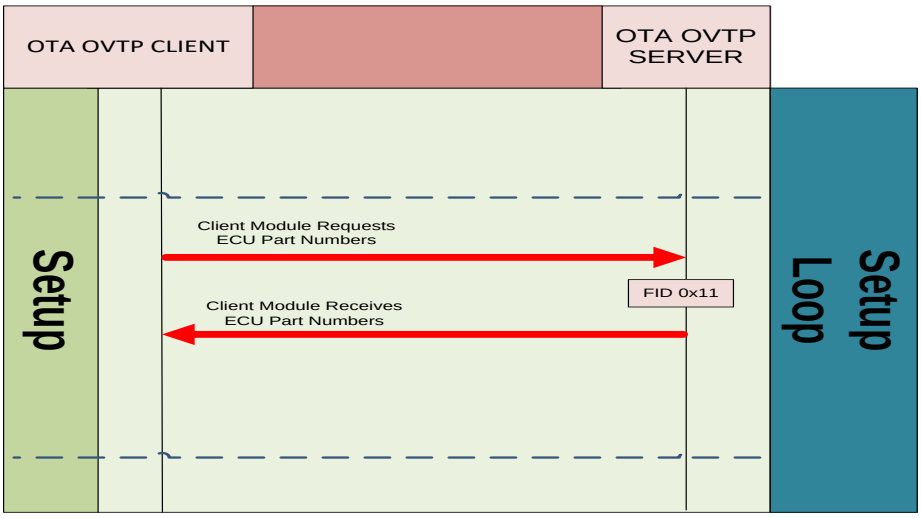


Figure 2: Example OTA Read DIDs Flowchart

REQ-308099/D-authorizeEraseMemory (0x12) Function

2.1.1.15 Function Description

The authorizeEraseMemory function is used to authorize the erasure of physical memory based upon client specified address and length pairs. The expectation is that the backend cloud shall send an authorizeEraseMemory command to the OTA client for the specific module that needs to be erased based on the SUCounter, memory addresses, and memory sizes. This command shall be signed by the backend and will allow the module to determine if it is a valid erase authorization.

authorizeEraseMemory is a category 2 signed OTA request (see REQ-333407) and is paired with the eraseMemory (0x13) FID. authorizeEraseMemory shall never cause the stored SUcounter to be incremented.

The memory address and the memory size parameters will be repeated based on the number of blocks to authorize for erasure. If one memory block is to be authorized for erasure, then only one memory address and one memory size are needed. If 2 memory blocks are to be authorized for erasure, then a pair of memory addresses and memory sizes will be added. For every memory block to be authorized for erasure, 8 bytes will be added (4 bytes for the memory address and 4 bytes for the memory size). The server shall verify the memory address and length pairs are valid prior to sending a positive response.

2.1.1.16 OVTP Header Information

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

Table 18 — authorizeEraseMemory Header Info

Message	Cntr	CryptoType	SSN
Request	0	0	1
Response	0	0	1

2.1.1.17 Request Message

Table 19 — Request message definition

A_Data byte	Parameter Name	Cvt	Byte Value	Mnemonic
#1	authorizeEraseMemoryFID	M	0x12	AEM
	FESN[] = [			FESN
#2	FESN#1 (MSB)	M	0x00 – 0xFF	B1
:	:	:	:	:
#9	FESN#8]	M	0x00 – 0xFF	B8
	SUCounter[] = [			SUC_
#10	Byte#1 (MSB)	M	0x00 – 0xFF	B1
#11	Byte#2	M	0x00 – 0xFF	B2
#12	Byte#3	M	0x00 – 0xFF	B3
#13	Byte#4]	M	0x00 – 0xFF	B4

A_Data byte	Parameter Name	Cvt	Byte Value	Mnemonic
	memoryAddress[] = [			MA_
#14	Byte#1 (MSB)	M	0x00 – 0xFF	B1
#15	Byte#2	M	0x00 – 0xFF	B2
#16	Byte#3	M	0x00 – 0xFF	B3
#17	Byte#4]	M	0x00 – 0xFF	B4
	memorySize[] = [			MS_
#18	Byte#1 (MSB)	M	0x00 – 0xFF	B1
#19	Byte#2	M	0x00 – 0xFF	B2
#20	Byte#3	M	0x00 – 0xFF	B3
#21	Byte#4]	M	0x00 – 0xFF	B4
:	:	:	:	:
	memoryAddress[] = [			MA_
#n - 263	Byte#1 (MSB)	C	0x00 – 0xFF	B1
#n - 262	Byte#2	C	0x00 – 0xFF	B2
#n - 261	Byte#3	C	0x00 – 0xFF	B3
#n - 260	Byte#4]	C	0x00 – 0xFF	B4
	memorySize[] = [			MS_
#n - 259	Byte#1 (MSB)	C	0x00 – 0xFF	B1
#n - 258	Byte#2	C	0x00 – 0xFF	B2
#n - 257	Byte#3	C	0x00 – 0xFF	B3
#n - 256	Byte#4]	C	0x00 – 0xFF	B4
	eraseSignature[] = [			ESIG
#n - 255	Byte#1 (MSB)	M	0x00 – 0xFF	B1
:	:	:	:	:
:	:	:	:	:
#n	Byte#256	M	0x00 – 0xFF	B256

Table 20 — Request message data-parameter definition

Definition
FESN The FORD ECU Serial Number is a unique identifier for the module being flashed
SUCounter The Software Update Counter, this counter is a 4 byte counter that is transmitted between the cloud and the module to ensure that any OTA authorization request is fresh. Refer to Ref [2] for more details.
memoryAddress This parameter is the starting address of the server memory that indicates a memory address and memory size that are authorized to be erased.
memorySize This parameter is the total number of bytes of memory authorized to be erased starting at the memoryAddress location.
eraseSignature This parameter is the secure command signature that ensures the request was authorized by Ford backend systems.

2.1.1.18 Positive Response Message

Table 21 — Positive Response message definition

A_Data byte	Parameter Name	Cvt	Byte Value	Mnemonic
#1	authorizeEraseMemory Response FID	M	0x92	AEMPR

2.1.1.19 Supported negative response codes

Table 22 — Supported negative response codes

NRC	Description	Mnemonic
0x13	**incorrectMessageLengthOrInvalidFormat** **This NRC shall be sent if the length of the request message is invalid.**	**IMLOIF**
0x15	**endToEndSignatureInvalid** **This NRC shall be sent if the received eraseSignature is not valid**	**ETESI**
0x16	**FESNinvalid** **This NRC shall be sent if the received FESN does not match the ECU's FESN.**	**FESNI**
0x17	**SUCounterInvalid** **This NRC shall be sent if the end to end signature was valid but the received SUCounter is lower than the expected internal SUCounter of the ECU**	**SUCI**
0x31	**requestOutOfRange** **This NRC shall be sent if:** **Any memory address within the interval [MA, MA + MS 1)] is invalid or restricted;** **The memorySize parameter value in the request message is not supported by the server;** **The memorySize parameter value in the request message is zero;**	**ROOR**

2.1.1.20 Message flow example(s)

Table 23 — authorizeEraseMemory 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	authorizeEraseMemory Request FID	0x12	AEM
2	FESN[1]	0x11	FESN
3	FESN[2]	0x22	FESN
4	FESN[3]	0x33	FESN
5	FESN[4]	0x44	FESN
6	FESN[5]	0x55	FESN
7	FESN[6]	0x66	FESN
8	FESN[7]	0x77	FESN
9	FESN[8]	0x88	FESN
10	SUCounter[1]	0x00	SUC
11	SUCounter[2]	0x00	SUC
12	SUCounter[3]	0x00	SUC
13	SUCounter[4]	0x01	SUC
14	Memory Address [1]	0x00	MA
15	Memory Address [2]	0x00	MA
16	Memory Address [3]	0x10	MA
17	Memory Address [4]	0x00	MA
18	Memory Size [1]	0x00	MS

Message direction	client → server		
Message Type	Request		
A_Data byte	Description (all values are in hexadecimal)	Byte Value	Mnemonic
19	Memory Size [2]	0x00	MS
20	Memory Size [3]	0x20	MS
21	Memory Size [4]	0x00	MS
22	Memory Address [1]	0x00	MA
23	Memory Address [2]	0x03	MA
24	Memory Address [3]	0x00	MA
25	Memory Address [4]	0x00	MA
26	Memory Size [1]	0x00	MS
27	Memory Size [2]	0x04	MS
28	Memory Size [3]	0x00	MS
29	Memory Size [4]	0x00	MS
30 - 285	eraseSignature	0xXX	ESIG

Table 24 — 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	authorizeEraseMemory Response FID	0x92	AEMPR

REQ-308100/E-eraseMemory (0x13) Function

2.1.1.21 Function Description

The eraseMemory function is used to erase physical memory based upon a client specific address and length. The server shall only provide a positive response after successfully completing erasure of all memory included in the request.

The eraseMemory Function call shall only be accepted if an authorizeEraseMemory (0x12) function call accepted during the current OVTP session is still authorized (see REQ-333407) and the requested memoryAddress and memorySize was included within an authorized pair in that preceding authorizeEraseMemory request. If the memoryAddress and memorySize are not authorized, the server shall not execute this function and shall reply with the defined NRC.

2.1.1.22 OVTP Header Information

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

Table 25 — eraseMemory Header Info

Message	Cntr	CryptoType	SSN
Request	0	0	1

Response	0	0	1

2.1.1.23 Request Message

Table 26 — Request message definition

A_Data byte	Parameter Name	Cvt	Byte Value	Mnemonic
#1	eraseMemoryFID	M	0x13	EM
	memoryAddress[] = [			MA_
#2	Byte#1 (MSB)	M	0x00 – 0xFF	B1
#3	Byte#2	M	0x00 – 0xFF	B2
#4	Byte#3	M	0x00 – 0xFF	B3
#5	Byte#4]	M	0x00 – 0xFF	B4
	memorySize[] = [			MS_
#6	Byte#1 (MSB)	M	0x00 – 0xFF	B1
#7	Byte#2	M	0x00 – 0xFF	B2
#8	Byte#3	M	0x00 – 0xFF	B3
#9	Byte#4]	M	0x00 – 0xFF	B4

Table 27 — Request message data-parameter definition

Definition
memoryAddress This parameter is the starting address of the server memory where the erase shall start at.
memorySize This parameter is the total number of bytes of memory to be erased starting at the memoryAddress location.

2.1.1.24 Positive Response Message

Table 28 — Response message definition

A_Data byte	Parameter Name	Cvt	Byte Value	Mnemonic
#1	Erase Memory Response FID	M	0x93	EMPR

2.1.1.25 Supported negative response codes

Table 29 — Supported negative response codes

NRC	Description	Mnemonic
0x13	**incorrectMessageLengthOrInvalidFormat** This NRC shall be sent if the length of the request message is invalid.	IMLOIF
0x20	**requestProcessingSuspendedRepeatRequest** This NRC indicates that the requested action will be suspended and the client must re-request to resume the action. A typical example usage would be when network is ready to sleep and OTA server needs to suspend the action.	RPSRR
0x22	**conditionsNotCorrect** This NRC indicates that the requested action will not be taken because the server prerequisite conditions are not met (e.g., voltage out of range).	CNC
0x31	**requestOutOfRange** This NRC shall be sent if the memoryAddress and memorySize combination do not align with the erasable flash sector.	ROOR

NRC	Description	Mnemonic
0x33	**securityRequired** This NRC indicates that the erase for the specific memory address and length are not currently authorized.	SR
0x72	**generalProgrammingFailure** This NRC indicates that the server detected an error when erasing or programming a memory location in the permanent memory device (e.g., flash memory).	GPF

2.1.1.26 Message flow example(s)

Table 30 — Erase Memory 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	eraseMemory Request FID	0x13	EM
2	Memory Address [1]	0x00	MA
3	Memory Address [2]	0x00	MA
4	Memory Address [3]	0x10	MA
5	Memory Address [4]	0x00	MA
6	Memory Size [1]	0x00	MS
7	Memory Size [2]	0x00	MS
8	Memory Size [3]	0x20	MS
9	Memory Size [4]	0x00	MS

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	Erase Memory Positive Response ID	0x93	EMPR

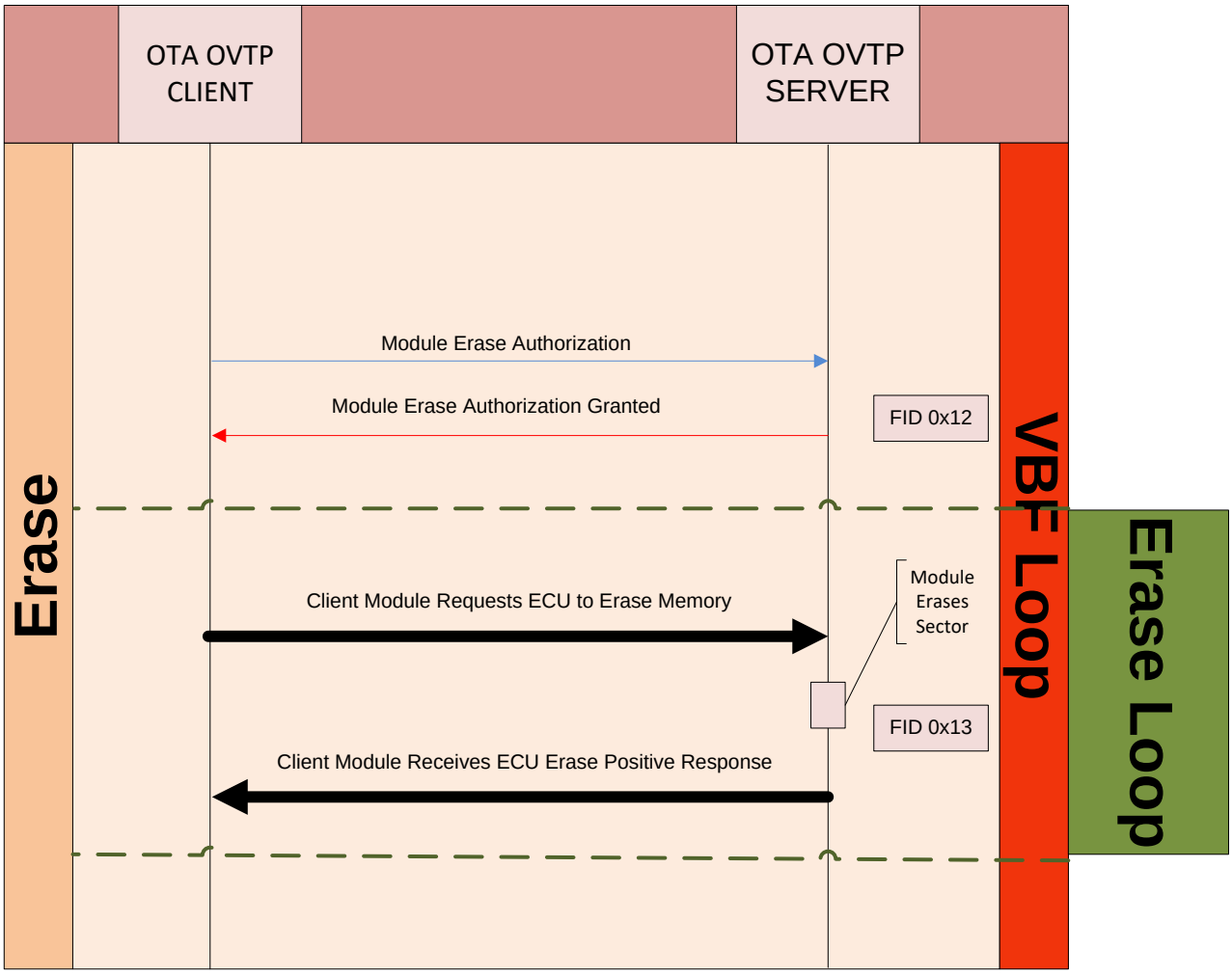


Figure 3: Example Erase Memory Flowchart

REQ-308101/D-authorizeDownload (0x14) Function

2.1.1.27 Function Description

The authorizeDownload function is used to authorize the programming of physical memory based upon client specified address and length pairs. The expectation is that the backend cloud shall send an authorizeDownload command to the OTA client for the specific module that needs to be programmed based on the SUCounter, memory addresses, and memory sizes. This command shall be signed by the backend and will allow the module to determine if it is a valid programming authorization.

authorizeDownload is a category 2 signed OTA request (see REQ-333407) and is paired with the initiateDownload (0x15) FID. The authorizeDownload shall never cause the stored SUcounter to be incremented.

The memory address and the memory size will be repeated based on the number of blocks to authorize for programming. If one memory block is to be authorized for programming, then only one memory address and one memory size are needed. If 2 memory blocks are to be authorized for programming, then a pair of memory address and memory size will be added. For every memory block to program, 8 bytes will be added (4 bytes for the memory address and 4 bytes for the memory size). The server shall verify the memory address and length pairs are valid prior to sending a positive response.

2.1.1.28 OVTP Header Information

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

Table 32 — authorizeDownload Header Info

Message	Cntr	CryptoType	SSN
Request	0	0	1
Response	0	0	1

2.1.1.29 Request Message

Table 33 — Request message definition

A_Data byte	Parameter Name	Cvt	Byte Value	Mnemonic
#1	authorizeDownload FID	M	0x14	AD
	FESN[] = [			FESN
#2	FESN#1 (MSB)	M	0x00 – 0xFF	B1
:	:	:	:	:
#9	FESN#8]	M	0x00 – 0xFF	B8
	SUCounter[] = [			CCC
#10	Byte#1 (MSB)	M	0x00 – 0xFF	B1
#11	Byte#2	M	0x00 – 0xFF	B2
#12	Byte#3	M	0x00 – 0xFF	B3
#13	Byte#4]	M	0x00 – 0xFF	B4

A_Data byte	Parameter Name	Cvt	Byte Value	Mnemonic
	memoryAddress#1[] = [			MA_
#14	Byte#1 (MSB)	M	0x00 – 0xFF	B1
#15	Byte#2	M	0x00 – 0xFF	B2
#16	Byte#3	M	0x00 – 0xFF	B3
#17	Byte#4]	M	0x00 – 0xFF	B4
	memorySize#1[] = [			MS_
#18	Byte#1 (MSB)	M	0x00 – 0xFF	B1
#19	Byte#2	M	0x00 – 0xFF	B2
#20	Byte#3	M	0x00 – 0xFF	B3
#21	Byte#4]	M	0x00 – 0xFF	B4
:	:	:	:	:
	memoryAddress#m[] = [			MA_
#22 + 8n	Byte#1 (MSB)	O	0x00 – 0xFF	B1
#23 + 8n	Byte#2	O	0x00 – 0xFF	B2
#24 + 8n	Byte#3	O	0x00 – 0xFF	B3
#25 + 8n	Byte#4]	O	0x00 – 0xFF	B4
	memorySize#m[] = [			MS_
#26 + 8n	Byte#1 (MSB)	O	0x00 – 0xFF	B1
#27+ 8n	Byte#2	O	0x00 – 0xFF	B2
#28+ 8n	Byte#3	O	0x00 – 0xFF	B3
#29 + 8n	Byte#4]	O	0x00 – 0xFF	B4
	downloadSignature[] = [			DSIG
#29 + 8n + 1	Byte#1 (MSB)	M	0x00 – 0xFF	B1
:	:	:	:	:
:	:	:	:	:
#29 + 8n + 256	Byte#256	M	0x00 – 0xFF	B256

Table 34 — Request message data-parameter definition

Definition
FESN The FORD ECU Serial Number is a unique identifier for the module being flashed
SUCounter The Software Update Counter, this counter is a 4 byte counter that is transmitted between the cloud and the module to ensure that any OTA authorization request is fresh. Refer to Ref [2] for more details.
memoryAddress This parameter is the starting address of the server memory where the allowed programming shall start at.
memorySize This parameter is the total number of bytes of memory to be allowed for programming, starting at the memoryAddress location.
downloadSignature This parameter is the secure command signature that ensures the request was authorized by Ford backend systems.

2.1.1.30 Positive Response Message

Table 35 — Response message definition

A_Data byte	Parameter Name	Cvt	Byte Value	Mnemonic
#1	authorizeDownload Response FID	M	0x94	ADPR

2.1.1.31 Supported negative response codes

Table 36 — Supported negative response codes

NRC	Description	Mnemonic
0x13	**incorrectMessageLengthOrInvalidFormat** **This NRC shall be sent if the length of the request message is invalid.**	**IMLOIF**
0x15	**endToEndSignatureInvalid** **This NRC indicates that an embedded end to end signature in the function was deemed invalid by the server.**	**ETESI**
0x16	**FESNInvalid** **This NRC indicates that an embedded FORD electronic serial number in the function was deemed invalid by the server.**	**FESNI**
0x17	**SUCounterInvalid** **This NRC shall be sent if the end to end signature was valid but the received SUCounter is lower than the expected internal SUCounter of the ECU**	**SUCI**
0x31	**requestOutOfRange** **This NRC shall be sent if:** **Any memory address within the interval [0xMA, (0xMA + 0xMS -0x1)] is invalid or restricted;** **The memorySize parameter value in the request message is not supported by the server;** **The memorySize parameter value in the request message is zero**	**ROOR**

2.1.1.32 Message flow example(s)

Table 37 — authorizeDownload 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	authorizeDownload Request FID	0x14	AD
2	FESN[1]	0x11	FESN
3	FESN[2]	0x22	FESN
4	FESN[3]	0x33	FESN
5	FESN[4]	0x44	FESN
6	FESN[5]	0x55	FESN
7	FESN[6]	0x66	FESN
8	FESN[7]	0x77	FESN
9	FESN[8]	0x88	FESN
10	SUCounter[1]	0x00	SUC
11	SUCounter[2]	0x00	SUC
12	SUCounter[3]	0x00	SUC
13	SUCounter[4]	0x02	SUC
14	Memory Address [1]	0x00	MA
15	Memory Address [2]	0x00	MA
16	Memory Address [3]	0x10	MA
17	Memory Address [4]	0x00	MA
18	Memory Size [1]	0x00	MS
19	Memory Size [2]	0x00	MS

Message direction	client → server		
Message Type	Request		
A_Data byte	Description (all values are in hexadecimal)	Byte Value	Mnemonic
20	Memory Size [3]	0x20	MS
21	Memory Size [4]	0x00	MS
22	Memory Address [1]	0x00	MA
23	Memory Address [2]	0x03	MA
24	Memory Address [3]	0x00	MA
25	Memory Address [4]	0x00	MA
26	Memory Size [1]	0x00	MS
27	Memory Size [2]	0x04	MS
28	Memory Size [3]	0x00	MS
29	Memory Size [4]	0x00	MS
30 - 285	downloadSignature	0xXX	DSIG

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	authorizeDownload Response FID	0x94	ADPR

REQ-308102/C-initiateDownload (0x15) Function

2.1.1.33 Function Description

The initiateDownload function is used to initiate an OTA over OVTP data transfer from the client to the server. After the server has received the initiateDownload request, the server shall take all necessary actions to receive data before it sends a positive response message.

If the dataFormatIdentifier in the initiateDownload request indicates the data is uncompressed and unencrypted, then the initiateDownload function shall only be accepted if an authorizeDownload (0x14) function call accepted during the current OVTP session is still authorized (see REQ-333407) and the requested memoryAddress and memorySize falls within an authorized pair in that preceding authorizeDownload request. If the memoryAddress and memorySize are not within an authorized range, the server shall not execute this function and shall reply with the defined NRC.

If the dataFormatIdentifier in the initiateDownload request indicates the data is compressed or encrypted, then the initiateDownload function shall only be accepted if an authorizeDownload (0x14) function call accepted during the current OVTP session is still authorized (see REQ-333407). In addition, the requested memoryAddress must fall within a logical block that has any authorized address in the preceding authorizeDownload request. No validation is required on the memorySize parameter to fall within an authorized pair in the preceding authorizeDownload request. These modified checks are necessary to account for two scenarios. The first scenario is when a pause / resume occurs and more bytes have been written (due to decompression) than the authorized compressed size. The second scenario is when poorly compressed data results in a block of data that is larger than the decompressed size.

2.1.1.34 OVTP Header Information

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

Table 39 — initiateDownload Header Info

Message	Cntr	CryptoType	SSN
Request	0	0	1
Response	0	0	1

2.1.1.35 Request Message

Table 40 — Request message definition

A_Data byte	Parameter Name	Cvt	Byte Value	Mnemonic
#1	initiateDownload FID	M	0x15	ID
#2	dataFormatdentfier	M	0x00-0xFF	DFI
	memoryAddress[] = [			MA_
#3	Byte#1 (MSB)	M	0x00 – 0xFF	B1
#4	Byte#2	M	0x00 – 0xFF	B2
#5	Byte#3	M	0x00 – 0xFF	B3
#6	Byte#4	M	0x00 – 0xFF	B4
	memorySize[] = [			MS_
#7	Byte#1 (MSB)	M	0x00 – 0xFF	B1
#8	Byte#2	M	0x00 – 0xFF	B2
#9	Byte#3	M	0x00 – 0xFF	B3
#10	Byte#4	M	0x00 – 0xFF	B4

Table 41 — Request message data-parameter definition

Definition
dataFormatIdentifier This data-parameter is a one byte value with each nibble encoded separately. The high nibble specifies the "compressionMethod", and the low nibble specifies the "encryptionMethod". The value 0x00 specifies that neither compressionMethod nor encryptionMethod is used. Values other than 0x00 are described in reference [4].
memoryAddress The parameter memoryAddress is the starting address of the server memory where the data is to be written to. The number of bytes used for this address is set to 4 bytes.
memorySize This parameter shall be used by the server to compare the memory size with the total amount of data transferred during the transferData function. The number of bytes used for this size is set to 4 bytes.

2.1.1.36 Positive Response Message

Table 42 — Positive Response message definition

A_Data byte	Parameter Name	Cvt	Byte Value	Mnemonic
#1	initiateDownload Response FID	M	0x95	IDPR

A_Data byte	Parameter Name	Cvt	Byte Value	Mnemonic
	maxNumberOfBlockLength = [			MNOBL_
#2	byte#1 (MSB)	M	0x00 – 0xFF	B1
#3	byte#2]	M	0x00 – 0xFF	B2

Table 43 — Positive response message data-parameter definition

Definition
maxNumberOfBlockLength This parameter is used by the initiateDownload positive response message to inform the client how many data bytes (maxNumberOfBlockLength) to include in each transferData request message from the client. This length reflects the actual size of the transferRequestParameterRecord in the transferData function. Note that the last transferData request within a given block may be required to be less than maxNumberOfBlockLength.

2.1.1.37 Supported negative response codes

Table 44 — Supported negative response codes

NRC	Description	Mnemonic
0x13	**incorrectMessageLengthOrInvalidFormat** **This NRC shall be sent if the length of the request message is invalid.**	**IMLOIF**
0x22	**conditionsNotCorrect** **This NRC shall be returned if a server receives another download request while a previous initiateDownload memory is being programmed** **This NRC indicates that the requested action will not be taken because the server prerequisite conditions are not met (e.g., voltage out of range).**	**CNC**
0x31	**requestOutOfRange** **the specified dataFormatIdentifier is not valid.** **The memorySize/memory address parameter value in the request message is not supported by the server;**	**ROOR**
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**
0x70	**downloadNotAccepted** **This NRC indicates than an attempt to download to a server's memory cannot be accomplished due to some fault conditions.** **This NRC shall be sent if an OTA Initiate Download Function in Progress is true (see DID 0xD022 parameter #1), and the memoryAddress is authorized, but does not equal the expected address from DID 0xD022 (OTA Last Address Successfully Written + 1).**	**DNA**

REQ-308103/D-transferData (0x16) Function

2.1.1.38 Function Description

The transferData function provides the ability to transfer from the client to the server in order to update the server memory. The request must be preceded by an initiateDownload request in the current OVTP session. The transferData function request includes a blockSequenceCounter to allow for improved error handling in case a transferData request fails during a sequence of multiple transferData requests. The blockSequenceCounter of the server shall be initialized to one upon positively responding to any initiateDownload request message. This means that the first transferData request message following an accepted initiateDownload request message shall utilize a blockSequenceCounter of one.

If a transferData request to download data is correctly received and processed in the server but the positive response message does not reach the client then the client would determine an application layer timeout and would repeat the same request (including the same blockSequenceCounter). The server would receive the repeated transferData request and could determine based on the included blockSequenceCounter that this transferData request is repeated. The server would send the positive response message immediately without writing the data once again into its memory.

If the transferData request to download data is not received correctly in the server then the server would not send a positive response message. The client would determine an application layer timeout and would repeat the same request (including the same blockSequenceCounter). The server would receive the repeated transferData request and could determine based on the included blockSequenceCounter that this is a new transferData request.

When a transferData request is received that completes the transferring of all information from the corresponding initiateDownload request, the server shall always complete any and all actions necessary to fully write all data into non-volatile memory prior to providing a positive response so that a completeDownload is not required to be received.

A transferData received during an active initiateDownload that results in a negative response shall not de-activate the initiateDownload.

2.1.1.39 OVTP Header Information

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

Table 45 — transferData Header Info

Message	Cntr	CryptoType	SSN
Request	0	0	1
Response	0	0	1

2.1.1.40 Request Message

Table 46 — Request message definition

A_Data byte	Parameter Name	Cvt	Byte Value	Mnemonic
#1	transferData FID	M	0x16	TD
#2	blockSequenceCounter	M	0x00 – 0xFF	BSC

A_Data byte	Parameter Name	Cvt	Byte Value	Mnemonic
	transferRequestParameterRecord[] = [			TRPR_
#3	transferRequestParameter#1	M	0x00 – 0xFF	TRTP_1
:	:	:	:	:
#n	transferRequestParameter#m]	U	0x00 – 0xFF	TRTP_m

Table 47 — Request message data-parameter definition

Definition
blockSequenceCounter The blockSequenceCounter parameter value starts at 0x01 with the first transferData request that follows the initiateDownload function. Its value is incremented by 1 for each subsequent transferData request. At the value of 0xFF the blockSequenceCounter rolls over and starts at 0x00 with the next transferData request message.
transferRequestParameterRecord This parameter records contains the actual payload of data bytes what will be written to the server's memory. It shall be the same size as defined in the positive response to the Initiate Download function (maxNumberOfBlockLength). Note that the last Transfer Data request sent may be smaller than maxNumberOfBlockLength.

2.1.1.41 Positive Response Message

Table 48 — Response message definition

A_Data byte	Parameter Name	Cvt	Byte Value	Mnemonic
#1	transferData Response FID	M	0x96	TDPR
#2	blockSequenceCounter	M	0x00 – 0xFF	BSC

Table 49 — Response message data-parameter definition

Definition
blockSequenceCounter This parameter is an echo of the blockSequenceCounter parameter from the request message.

2.1.1.42 Supported negative response codes

Table 50 — Supported negative response codes

NRC	Description	Mnemonic
0x13	**incorrectMessageLengthOrInvalidFormat** **This NRC shall be sent if the length of the request message is invalid.**	**IMLOIF**
0x22	**conditionsNotCorrect** **This NRC indicates that the requested action will not be taken because the server prerequisite conditions are not met (e.g., voltage out of range).**	**CNC**
0x24	**requestSequenceError** **The server shall use this response code:** **If the initiateDownload service is not active when a request for this function is received;** **If the initiateDownload service is active, but the server has already received all data as determined by the memorySize parameter in the active initiateDownload function;** **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.**	**RSE**
0x72	**generalProgrammingFailure** **This NRC indicates that the server detected an error when erasing or programming a memory location in the device.**	**PF**
0x73	**wrongSequenceCounter** **This NRC indicates that the server received an invalid blockSequenceCounter value**	**RSV**

2.1.1.43 transferData Early Acknowledge Strategy

In order to optimize the bus utilization (especially when downloading to a single ECU), ECUs shall implement an "Early Acknowledge" strategy for transferData (FID 0x16). The "Early Acknowledgement" strategy is intended to better utilize the bus bandwidth. The ECU shall always send a positive response to a transferData request prior to starting the writing of data in the request to memory (with the exception of the final transferData request). The first transferData request within a download block shall result in a positive response being queued for transmit prior to starting the data write. This allows the ECU to utilize the time in which the next transferData request is being transmitted over the network to actually write the previous transferData to memory and therefore more effectively utilize the bus bandwidth. The ECU shall only provide a positive response to a subsequent transferData request within a given block after the data from the previous transferData request is confirmed to be successfully written to memory. The ECU shall not transmit a positive response to the final transferData request within a given block until the data from both the final request and the preceding request has been confirmed to be successfully written. Note that the above requirements regarding a positive response assume the standard transferData checks including expected blockSequenceCounter, message length, etc. are all correct. Using this approach, a negative response to a validly formatted transferData request normally indicates a problem writing data from the previous transferData request. Exceptions to implementing the "Early Acknowledge" strategy are allowed but only when explicitly approved by Ford Connected Vehicle & Services.
A simplified example illustrating the difference between the traditional data transfer (i.e., ECU writes all data prior to responding) and the "Early Acknowledge" strategy while sending a block of data consisting of four transferData requests is shown below. Transmit Time is the time required to send each transferData (FID 0x16) request. Program Time is the time required for the ECU to write the data from each transferData (FID 0x16) request to memory. For the example, the Transmit Time and Program Time are both assumed to be 100ms.

When “no Early Acknowledgement” the traditional programming implementation involves the ECU completely receiving and programming the data of each request prior to providing the positive response so that the tester may send the next request. Overall programming time for fully sending and receiving responses for the four transferData requests is 800ms.

Transmit Time #1	Bus Idle	Transmit Time #2	Bus Idle	Transmit Time #3	Bus Idle	Transmit Time #4	Bus Idle	
ECU Idle	Program Time #1	ECU Idle	Program Time #2	ECU Idle	Program Time #3	ECU Idle	Program Time #4	
0	100	200	300	400	500	600	700	800

The "Early Acknowledge" programming implementation involves the ECU providing a positive response after it has programmed the data from the previous request. Overall programming time for fully sending and receiving responses for the four transferData requests is 500ms.

Transmit Time #1	Transmit Time #2	Transmit Time #3	Transmit Time #4	Bus Idle	
	Program Time #1	Program Time #2	Program Time #3	Program Time #4	
0	100	200	300	400	500

The examples above assume the Transmit Time and Program Time are equal, but this is likely not the case. With the traditional programming implementation, the time to fully transmit and receive responses for N number of requests is equal to (N x Transmit Time) + (N x Program Time). With the "Early Acknowledge" implementation, the time to fully transmit and receive responses for N number of requests is reduced to (N x (Larger{Transmit Time, Program Time})) + Smaller{Transmit Time, Program Time}. Alternatively, the savings from implementing the "Early Acknowledge" strategy is equal to (N – 1) x (Smaller{Transmit Time, Program Time}).

2.1.1.44 Message flow example(s)

Table 51 — Transfer Data 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	transferData FID	0x16	TD
2	blockSequenceCounter	0x01	BSC
3	transferRequestParameter#1	0x11	TRTP_1
4	transferRequestParameter#2	0x12	TRTP_2
5	transferRequestParameter#3	0x13	TRTP_3
6	transferRequestParameter#4	0x14	TRTP_4
8	transferRequestParameter#5	0x15	TRTP_5
9	transferRequestParameter#6	0x16	TRTP_6
10	transferRequestParameter#7	0x17	TRTP_7
11	transferRequestParameter#8	0x18	TRTP_8

Table 52 — 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	transferData Response FID	0x96	TDPR
2	blockSequenceCounter	0x01	BSC

REQ-308104/C-completeDownload (0x17) Function

2.1.1.45 Function Description

The completeDownload function is used by the client in order to terminate an existing data transfer (based upon an active initiateDownload request).

When the completeDownload function is received, it shall provide a positive response only if an initiateDownload is currently active in the current OVTP session (i.e., an initiateDownload was received and not been terminated with a completeDownload) and all data specified in the active initiateDownload has been successfully transferred and written using transferData functions. As the OTA download process may be interrupted and span ignition cycles, if the server does not receive the completeDownload after receiving all data specified from an initiateDownload function, this shall not prevent the ECU from receiving a new initiateDownload request nor prevent the overall OTA programming process. For example, it is possible the client sends all required data for an initiateDownload request, but the customer keys off the vehicle thereby preventing a completeDownload from being sent. When the new OVTP session is opened, because the transfer of all data has already been completed, the client would not send a completeDownload request.

A completeDownload received during an active initiateDownload that results in a negative response shall not de-activate the initiateDownload.

2.1.1.46 OVTP Header Information

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

Table 53— completeDownload Header Info

Message	Cntr	CryptoType	SSN
Request	0	0	1
Response	0	0	1

2.1.1.47 Request Message

Table 54 — Request message definition

A_Data_byte	Parameter Name	Cvt	Byte Value	Mnemonic
#1	completeDownload Request FID	M	0x17	CD

2.1.1.48 Positive Response Message

Table 55 — Positive response message definition

A_Data byte	Parameter Name	Cvt	Byte Value	Mnemonic
#1	completeDownload Response FID	M	0x97	CDPR

2.1.1.49 Supported negative response codes

Table 56 — Supported negative response codes

NRC	Description	Mnemonic
0x13	**incorrectMessageLengthOrInvalidFormat** **This NRC shall be sent if the length of the request message is invalid.**	**IMLOIF**
0x24	**requestSequenceError** **This NRC shall be returned if:** **The programming process is not completed when a request for this function is received** **The initiateDownload function is not active.**	**RSE**

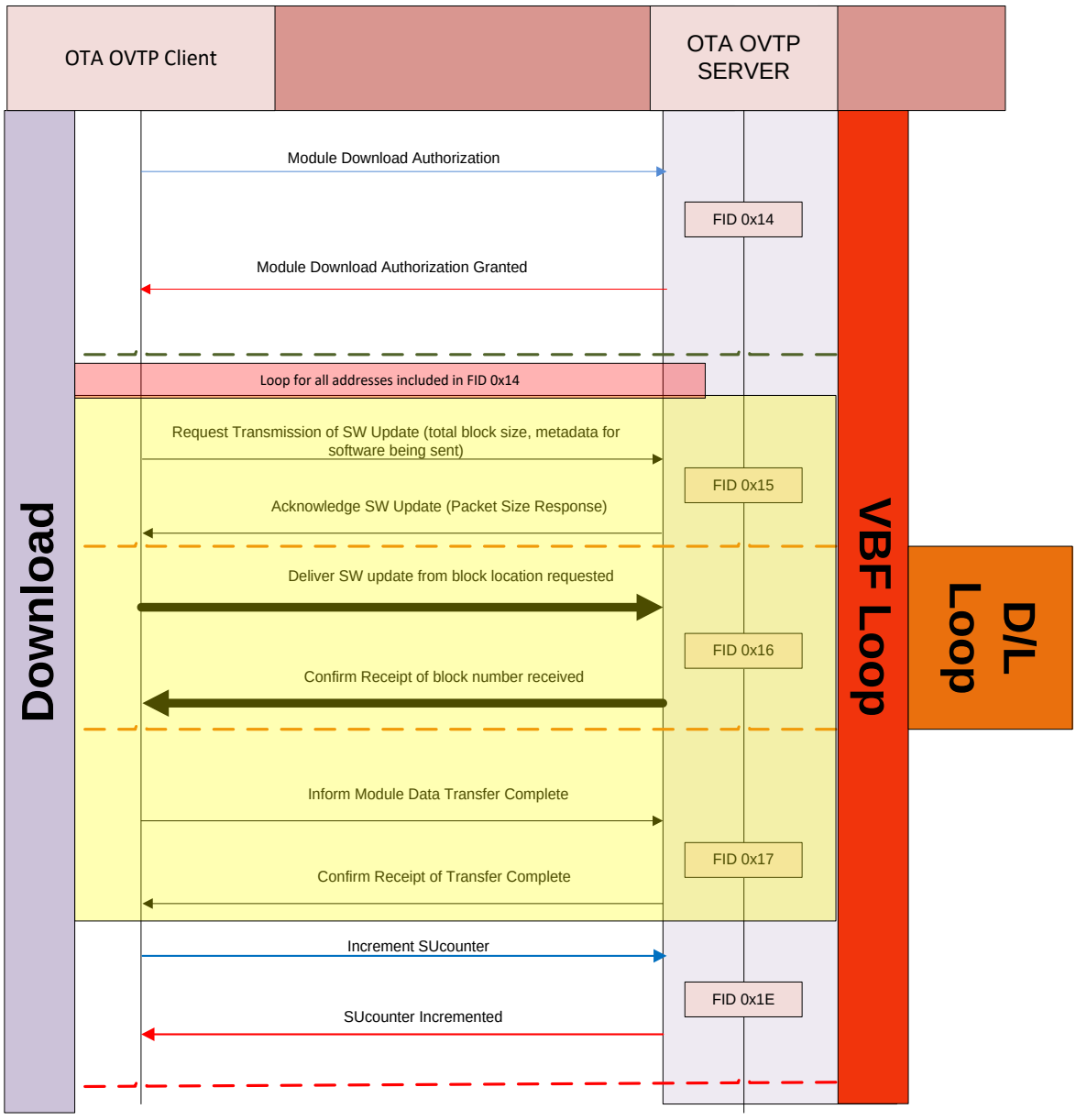


Figure 4: Example Download Flowchart

REQ-308105/D-diffUpdate (0x18) Function

2.1.1.50 Function Description

The diffUpdate function provides the request to the server to unpack and process a differential file.

diffUpdate is a category 1 signed OTA request (see REQ-333407). If the request is deemed valid as described in REQ-333407, the server shall validate the verificationStructureAddress correlates to an address designated to correspond to a differential file. If the above is valid, the server shall at minimum perform the following actions prior to providing a positive response:

- Verify the logical block referenced by the verificationStructureAddress is properly signed
- Unpack the differential file to the appropriate inactive memory

Note that the signature validation of the differential file's corresponding inactive memory partition is validated with a separate validateLogicalBlock request.

2.1.1.51 OVTP Header Information

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

Table 57 — diffUpdate Header Info

Message	Cntr	CryptoType	SSN
Request	0	0	1
Response	0	0	1

2.1.1.52 Request Message

Table 58 — Request message definition

A_Data byte	Parameter Name	Cvt	Byte Value	Mnemonic
#1	diffUpdate Request FID	M	0x18	DU
	FESN[] = [			FESN
#2	FESN#1 (MSB)	M	0x00 – 0xFF	B1
:	:	:	:	:
#9	FESN#8]	M	0x00 – 0xFF	B8
	SUCounter[] = [			SUC_
#10	Byte#1 (MSB)	M	0x00 – 0xFF	B1
#11	Byte#2	M	0x00 – 0xFF	B2
#12	Byte#3	M	0x00 – 0xFF	B3
#13	Byte#4]	M	0x00 – 0xFF	B4
	verificationStructureAddress[] = [			MA_
#14	Byte#1 (MSB)	M	0x00 – 0xFF	B1
#15	Byte#2	M	0x00 – 0xFF	B2
#16	Byte#3	M	0x00 – 0xFF	B3
#17	Byte#4]	M	0x00 – 0xFF	B4

A_Data byte	Parameter Name	Cvt	Byte Value	Mnemonic
	diffSignature[] = [			DSIG
#18	Byte#1 (MSB)	M	0x00 – 0xFF	B1
:	:	:	:	:
:	:	:	:	:
#273	Byte#256	M	0x00 – 0xFF	B256

Table 59 — Request message parameter definition

Definition
FESN The ECU FORD Serial Number is a unique identifier for the module being flashed
SUCounter The Software Update Counter, this counter is a 4 byte counter is transmitted between the cloud and the module to ensure that any OTA authorization request is fresh. More details in reference [2].
diffSignature This parameter is the secure command signature that ensures the request was authorized by Ford backend systems.
verificationStructureAddress The verificationStructureAddress contains the beginning address of the verification structure of the diff package.

2.1.1.53 Positive Response Message

Table 60 — Positive response message definition

A_Data byte	Parameter Name	Cvt	Byte Value	Mnemonic
#1	diffUpdate Response FID	M	0x98	DUPR

2.1.1.54 Supported negative response codes

Table 61 — Supported negative response codes

NRC	Description	Mnemonic
0x13	**incorrectMessageLengthOrInvalidFormat** This NRC shall be sent if the length of the request message is invalid.	IMLOIF
0x15	**diffSignatureInvalid** This NRC indicates that an embedded end to end signature in the function was deemed invalid by the server.	ETESI
0x16	**FESNInvalid** This NRC indicates that an embedded FORD electronic serial number in the function was deemed invalid by the server.	FESNI
0x17	**softwareUpdateCounterInvalid** This NRC indicates that an embedded software update counter in the function was deemed invalid by the server.	SUCI
0x20	**requestProcessingSuspendedRepeatRequest** This NRC indicates that the requested action will be suspended and the client must re-request to resume the action. A typical example usage would be when network is ready to sleep and OTA server needs to suspend the action.	RPSRR

NRC	Description	Mnemonic
0x22	**conditionsNotCorrect** This NRC indicates that the requested action will not be taken because the server prerequisite conditions are not met (e.g., voltage out of range).	CNC
0x24	**requestSequenceError** This NRC shall be sent if the diffUpdate functionality is unable to occur due to an active initiateDownload in progress.	RSE
0x31	**requestOutOfRange** This NRC shall be sent if the verificationStructureAddress is not a valid or supported address that corresponds to a differential memory area.	ROOR
0x72	**generalProgrammingFailure** This NRC indicates that the server detected an error when erasing or programming a memory location.	GPF
0x79	**validationFailed** This NRC shall be sent if the requested verificationStructureAddress does not contain a validly signed logical block.	VF

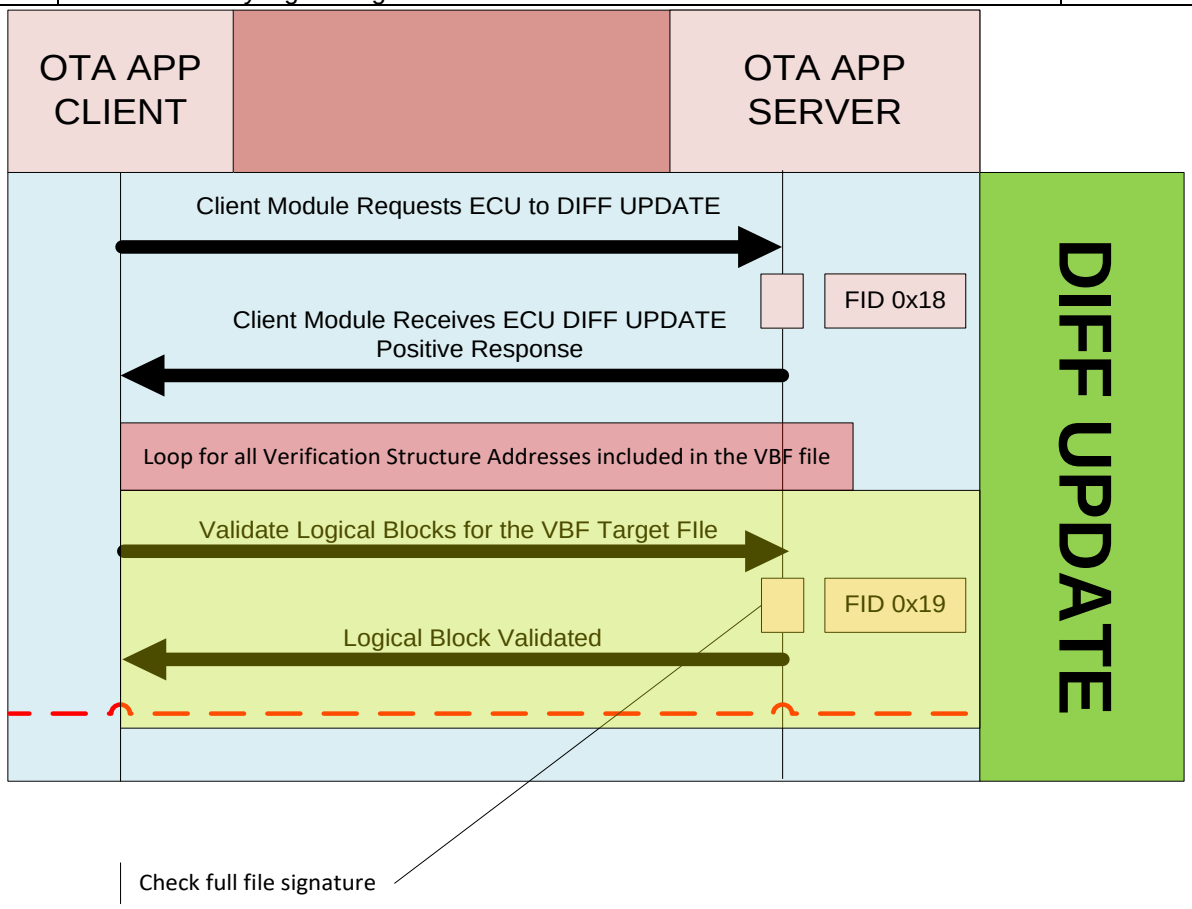


Figure 5: Example Diff Update Flowchart

REQ-308106/D-validateLogicalBlock (0x19) Function

2.1.1.55 Function Description

The validateLogicalBlock function provides the request for the server to perform the signature check on the software verification structure located at the client provided address. This allows the client to detect if information was not correctly writtne to memory for any reason before activation and to confirm that the server has verified the authenticity of this signed portion of software prior to activating memory.

FID 0x19 will be sent for every verification structure address that is being programmed. For example, FID 0x19 will be sent twice if there are 2 verification structure addresses.

2.1.1.56 OVTP Header Information

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

Table 62 — validateLogicalBlock Header Info

Message	Cntr	CryptoType	SSN
Request	0	0	1
Response	0	0	1

2.1.1.57 Request Message

Table 63 — Request message definition

A_Data byte	Parameter Name	Cvt	Byte Value	Mnemonic
#1	validateLogicalBlock Request FID	M	0x19	VLB
#2	verificationStructureAddress[] = [			VSA_
#3	Byte#1 (MSB)	M	0x00 – 0xFF	B1
#4	Byte#2	M	0x00 – 0xFF	B2
#5	Byte#3	M	0x00 – 0xFF	B3
	Byte#4]	M	0x00 – 0xFF	B4

Table 64 — Request message parameter definition

Definition
verificationStructureAddress The verificationStructureAddress contains the beginning address of the verification structure.

2.1.1.58 Positive Response Message

Table 65 — Positive response message definition

A_Data byte	Parameter Name	Cvt	Byte Value	Mnemonic
#1	validateLogicalBlock Response FID	M	0x99	VLBPR

A_Data byte	Parameter Name	Cvt	Byte Value	Mnemonic
	rootHash[] = [			RH
#2	rootHash#1 (MSB)	M	0x00 – 0xFF	B1
:	:	:	:	:
#33	rootHash#32]	M	0x00 – 0xFF	B32

Table 66 — Response message data-parameter definition

Definition
rootHash The hash calculated over a single verification structure used to generate the digital signature

2.1.1.59 Supported negative response codes

Table 67 — Supported negative response codes

NRC	Description	Mnemonic
0x13	**incorrectMessageLengthOrInvalidFormat** This NRC shall be sent if the length of the request message is invalid.	IMLOIF
0x20	**requestProcessingSuspendedRepeatRequest** This NRC indicates that the requested action will be suspended and the client must re-request to resume the action. A typical example usage would be when network is ready to sleep and OTA server needs to suspend the action.	RPSRR
0x24	**requestSequenceError** This NRC shall be sent if the validity of the logical block is unable to be checked due to an active initiateDownload in progress.	RSE
0x31	**requestOutOfRange** This NRC shall be sent if the verificationStructureAddress is not valid or supported by the server.	ROOR
0x79	**validationFailed** This NRC shall be sent if the requested verificationStructureAddress does not contain a validly signed logical block.	VF

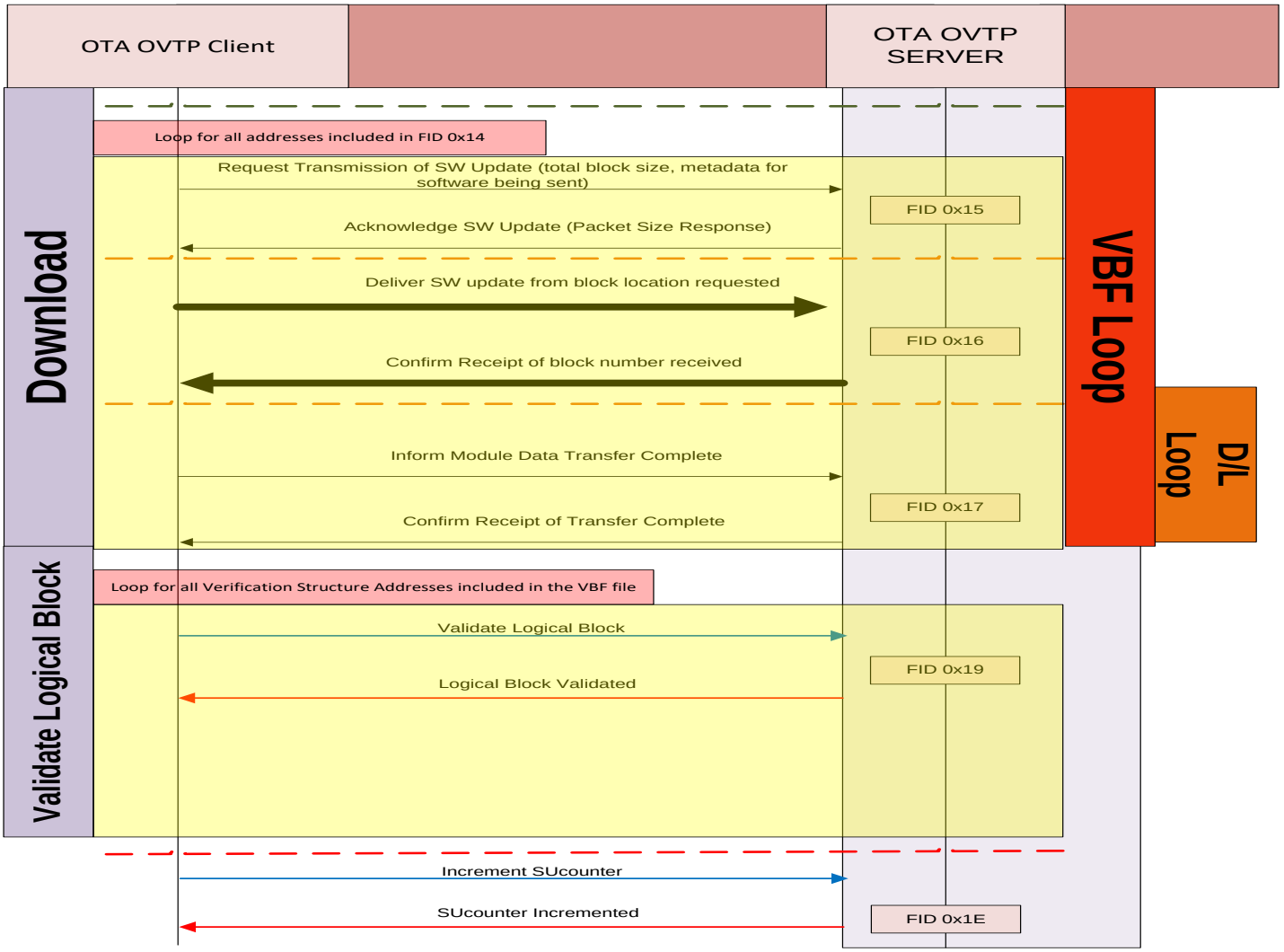


Figure 6: Validate Logical Block

REQ-308107/E-prepareActivation (0x1A) Function

2.1.1.60 Function Description

The prepareActivation function shall perform remaining operations needed to activate the new software in the inactive partition. Depending on the ECU architecture, these operations may involve backing up the active partition and/or copying any active logical blocks necessary to ensure a complete software is available to swap to. The prepareActivation request shall contain the complete list of verificationStructureAddresses supported by the server, excluding those which correspond to differential memory areas (if differentials are supported).

prepareActivation is a category 1 signed OTA request (see REQ-333407). If the request is deemed valid as described in REQ-333407, the server shall validate every single verificationStructureAddress it supports (excluding those corresponding to differential areas) is included in the request with no additional verificationStructureAddresses being present. If a given

verificationStructureAddress in the request does not have a validated logical block in the memory being used to swap to (or that logical block is valid but has not been updated since the last successful swap), the target ECU must copy that logical block from the active memory to the inactive memory if necessary for the ECU to swap to the inactive partition.

For example if X out of Y logical blocks have not been updated in the inactive partition since the last successful swap, upon receiving a valid prepareActivation request the server would do the following. If these X logical blocks in the inactive partition are not valid, or they are valid but their root hash does not match the corresponding logical block in the active partition, then:

1- Erase the corresponding logical blocks in the inactive memory
2- Copy the logical blocks from the active partition to the inactive partition
3- Include these transferred logical blocks in the SWash calculation (see reference [2])

The SWash calculation shall use the order of the verification structure addresses as specified in Appendix D. The SWash is used as a mechanism to verify the complete set of software that will be activated matches what is expected at the client.

If the ECU architecture supports OTA active back up, the ECU shall verify the active application and the OTA Active back are the same. If they are not the same , the ECU shall erase the active backup flash and copy active application into active backup flash (for use in rollback).

2.1.1.61 OVTP Header Information

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

Table 68 — prepareActivation Header Info

Message	Cntr	CryptoType	SSN
Request	0	0	1
Response	0	0	1

2.1.1.62 Request Message

Table 69 — Request message definition

A_Data byte	Parameter Name	Cvt	Byte Value	Mnemonic
#1	prepareActivation FID	M	0x1A	PATR
	FESN[] = [			FESN
#2	FESN#1 (MSB)	M	0x00 – 0xFF	B1
:	:	:	:	:
#9	FESN#8]	M	0x00 – 0xFF	B9
	SUCounter[] = [			SUC_
#10	Byte#1 (MSB)	M	0x00 – 0xFF	B1
#11	Byte#2	M	0x00 – 0xFF	B2
#12	Byte#3	M	0x00 – 0xFF	B3
#13	Byte#4]	M	0x00 – 0xFF	B4
	verificationStructureAddress[] = [			VSA_
#14	Byte#1 (MSB)	M	0x00 – 0xFF	B1
#15	Byte#2	M	0x00 – 0xFF	B2
#16	Byte#3	M	0x00 – 0xFF	B3
#17	Byte#4]	M	0x00 – 0xFF	B4
:	:	:	:	:

A_Data byte	Parameter Name	Cvt	Byte Value	Mnemonic
	verificationStructureAddress[] = [			VSA_
#n-291	Byte#1 (MSB)	O	0x00 – 0xFF	B1
#n-290	Byte#2	O	0x00 – 0xFF	B2
#n-289	Byte#3	O	0x00 – 0xFF	B3
#n-288	Byte#4]	O	0x00 – 0xFF	B4
	SWash[] = [			SWASH
#n-287	SWash#1 (MSB)	M	0x00 – 0xFF	B1
	:	:	:	:
#n-256	SWash#32]	M	0x00 – 0xFF	B32
	prepareActivationSignature[] = [			PASIG
#n - 255	Byte#1 (MSB)	M	0x00 – 0xFF	B1
:	:	:	:	:
:	:	:	:	:
#n	Byte#256	M	0x00 – 0xFF	B256

Table 70 — Request message parameter definition

Definition
FESN The ECU FORD Serial Number is a unique identifier for the module being flashed
SUCounter The Software Update Counter, this counter is a 4 byte counter that is transmitted between the cloud and the module to ensure that any OTA authorization request is fresh. Refer to Ref [2] for more details.
verificationStructureAddress The verificationStructureAddress contains the beginning address of the verification structure when the ECU implements software signing.
SWash The Software Hash is a hashing function of all the data partitions being updated by OTA (Refer to Appendix D)
prepareActivationSignature This parameter is the secure command signature that ensures the request was authorized by Ford backend systems.

2.1.1.63 Positive Response Message

Table 71 — Positive response message definition

A_Data byte	Parameter Name	Cvt	Byte Value	Mnemonic
#1	prepareActivation Response FID	M	0x9A	PATRPR

2.1.1.64 Supported negative response codes

Table 72 — Supported negative response codes

NRC	Description	Mnemonic
0x13	**incorrectMessageLengthOrInvalidFormat** This NRC shall be sent if the length of the request message is invalid.	IMLOIF
0x15	**endToEndSignatureInvalid** This NRC indicates that an embedded end to end signature in the function was deemed invalid by the server.	ETESI

NRC	Description	Mnemonic
0x16	**FESNInvalid** This NRC indicates that an embedded FORD electronic serial number in the function was deemed invalid by the server.	FESNI
0x17	**softwareUpdateCounterInvalid** This NRC indicates that an embedded software update counter in the function was deemed invalid by the server.	SUCI
0x20	**requestProcessingSuspendedRepeatRequest** This NRC indicates that the requested action will be suspended and the client must re-request to resume the action. A typical example usage would be when network is ready to sleep and OTA server needs to suspend the action.	RPSRR
0x22	**conditionsNotCorrect** This NRC indicates that the requested action will not be taken because the server prerequisite conditions are not met (e.g., voltage out of range)	CNC
0x31	**requestOutOfRange** This NRC shall be sent if one or more of the included verificationStructureAddress is not valid or supported by the server.	ROOR
0x72	**generalProgrammingFailure** This NRC indicates that the server detected an error when erasing or programming a memory location. This failure happens when the server is trying to copy some logical blocks from active memory to inactive memory (The server does this when the SWash request includes memory addresses for these blocks that were not downloaded before)	GPF
0xF8	**IncompatibleSoftware** This NRC indicates that the server detected incompatibility within different SW components (Say for example, Strategy and Cal-Configuration are not compatible). Note: This compatible check is in addition to Swash check. This is module specific requirement. If Target ECU (say for example, ABS) have specific requirement to perform this additional check, then, this NRC shall be supported.	IS
0x79	**validationFailed** This NRC shall be sent if the provided SWash does not match the ECU's calculated SWash.	VF

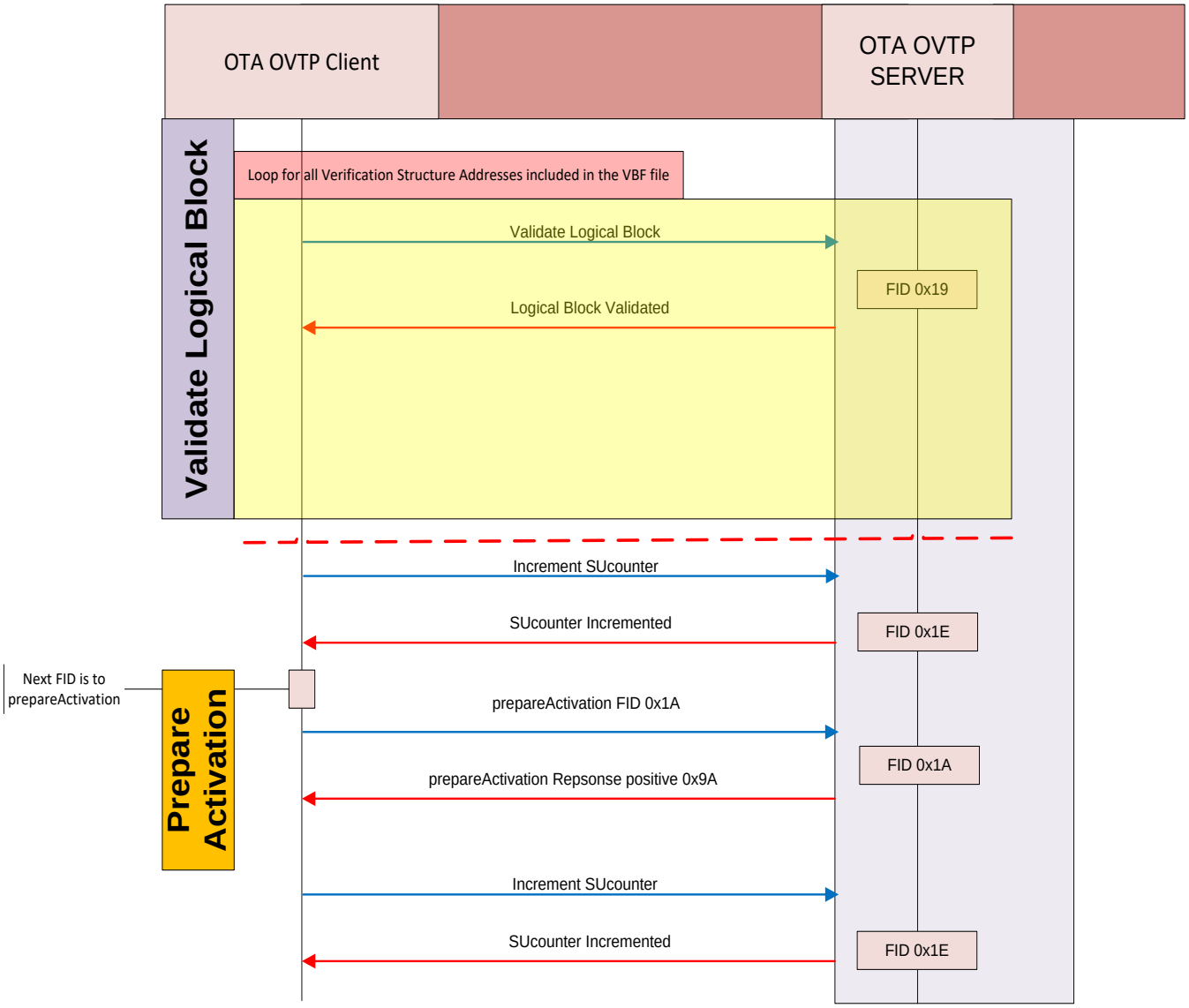


Figure 7: Example prepareActivation FID flowchart

REQ-308108/D-authorizeActivation (0x1B) Function

2.1.1.65 Function Description

The authorizeActivation function provides the means for the client to authorize the initiation of a trigger for activating the module's inactive application to become the active application. This assumes that all previous checks and erase, and download operations have been completed successfully.

authorizeActivation is a category 2 signed OTA request (see REQ-333407) and is paired with the initiateActivation (0x1C) FID. The authorizeActivation shall never cause the stored SUcounter to be incremented.

The authorizeActivation function shall include **all** verificationStructureAddresses used to compute the SWash in order to be accepted by the server. The SWash calculation shall use the order of the verification structure addresses as specified in Appendix D. The SWash is used as a mechanism to verify the complete set of software that will be activated matches what is expected at the client. The calculated SWash over what will become the active application if an initiateActivation is received must match the SWash in the request in order for this request to be accepted by the server.

If the ECU architecture supports OTA active back up, the ECU shall verify the active application and the OTA Active back are the same before erasing and copying from external to internal memory.

2.1.1.66 OVTP Header Information

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

Table 73 — authorizeActivation Header Info

Message	Cntr	CryptoType	SSN
Request	0	0	1
Response	0	0	1

2.1.1.67 Request Message

Table 74 — Request message definition

A_Data byte	Parameter Name	Cvt	Byte Value	Mnemonic
#1	authorizeActivation FID	M	0x1B	AA
	FESN[] = [			FESN
#2	FESN#1 (MSB)	M	0x00 – 0xFF	B1
:	:	:	:	:
#9	FESN#8]	M	0x00 – 0xFF	B8
	SUCounter[] = [			SUC_
#10	Byte#1 (MSB)	M	0x00 – 0xFF	B1
#11	Byte#2	M	0x00 – 0xFF	B2
#12	Byte#3	M	0x00 – 0xFF	B3
#13	Byte#4]	M	0x00 – 0xFF	B4
#14	triggerType	M	0x00 – 0xFF	TT
	verificationStructureAddress[] = [			VSA_
#15	Byte#1 (MSB)	M	0x00 – 0xFF	B1
#16	Byte#2	M	0x00 – 0xFF	B2
#17	Byte#3	M	0x00 – 0xFF	B3
#18	Byte#4]	M	0x00 – 0xFF	B4
:	:	:	:	:
	verificationStructureAddress[] = [			VSA_
#n-291	Byte#1 (MSB)	M	0x00 – 0xFF	B1
#n-290	Byte#2	M	0x00 – 0xFF	B2
#n-289	Byte#3	M	0x00 – 0xFF	B3
#n-288	Byte#4]	M	0x00 – 0xFF	B4
	SWash[] = [			SWASH
#n-287	SWash#1 (MSB)	M	0x00 – 0xFF	B1
:	:	:	:	:
#n-256	SWash#32]	M	0x00 – 0xFF	B32

A_Data byte	Parameter Name	Cvt	Byte Value	Mnemonic
	activationSignature[] = [			ASIG
#n-255	Byte#1 (MSB)	M	0x00 – 0xFF	B1
:	:	:	:	:
:	:	:	:	:
#n	Byte#256	M	0x00 – 0xFF	B256

Table 75 — Request message data-parameter definition

Definition
FESN The ECU FORD Serial Number is a unique numeric identifier for the module being flashed
SUCounter This parameter is a 4 byte counter that is transmitted between the cloud and the module to ensure that any authorized OTA request is fresh.
triggerType triggerType defines what action the ECU will take when receiving FID 0x1C (initiateActivation). States include: 0x00 : Immediate 0x01 – 0xFF : Reserved by document
verificationStructureAddress The verificationStructureAddress contains the beginning address of a verification structure.
SWash The Software Hash is a hashing function of all the data partitions being updated by OTA (Refer to Appendix D)
activationSignature This parameter is the secure command signature that ensures the request was authorized by Ford backend systems.

2.1.1.68 Positive Response Message

Table 76 — Request message definition

A_Data byte	Parameter Name	Cvt	Byte Value	Mnemonic
#1	authorizeActivation Response FID	M	0x9B	AAPR

2.1.1.69 Supported negative response codes

Table 77 — Supported negative response codes

NRC	Description	Mnemonic
0x13	**incorrectMessageLengthOrInvalidFormat** This NRC shall be sent if the length of the request message is invalid.	IMLOIF
0x16	**FESNInvalid** This NRC indicates that an embedded FORD electronic serial number in the function was deemed invalid by the server.	FESNI
0x15	**endToEndSignatureInvalid** This NRC indicates that an embedded end to end signature in the function was deemed invalid by the server.	ETESI
0x17	**softwareUpdateCounterInvalid** This NRC indicates that an embedded software update counter in the function was deemed invalid by the server.	SUCI

NRC	Description	Mnemonic
0x22	**conditionsNotCorrect** This NRC indicates that the requested action will not be taken because the server prerequisite conditions are not met (e.g., Vehicle is not in safe state)	CNC
0x31	**requestOutOfRange** This NRC shall be sent if triggerType or verificationStructureAddress is not valid or supported by the server	ROOR
0x72	**generalProgrammingFailure** This NRC indicates that all logical blocks in the inactive partition are not currently marked as valid	GPF
0xF8	**IncompatibleSoftware** This NRC indicates that the server detected incompatibility within different SW components (Say for example, Strategy and Cal-Configuration are not compatible). Note: This compatible check is in addition to Swash check. This is module specific requirement. If Target ECU (say for example, ABS) have specific requirement to perform this additional check, then, this NRC shall be supported.	IS
0x79	**validationFailed** This NRC shall be sent if the provided SWash does not match the ECU's calculated SWash.	VF

REQ-308109/E-initiateActivation (0x1C) Function

2.1.1.70 Function Description

The initiateActivation function provides the means for the client to request the ECU perform the actual swap to the inactive application so that it becomes active.

The initiateActivation function shall only be accepted if an authorizeActivation (0x1B) function call accepted during the current OVTP session is still authorized (see section REQ-333407). Upon accepting this request via a positive response, the server shall perform the activation of the inactive memory when the trigger occurs and perform any necessary actions (e.g., reset) for the currently inactive application to become active. After accepting this request by sending a positive response, the server shall not acknowledge or process any new OTA FID requests until the new software is fully activated, or an error has occurred preventing the activation.

In case of activation failure, the OTA server shall restore the OTA active backup (refer to Appendix A for implementation details).

2.1.1.71 OVTP Header Information

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

Table 78 — initiateActivation Header Info

Message	Cntr	CryptoType	SSN
Request	0	0	1
Response	0	0	1

2.1.1.72 Request Message

Table 79 — Request message definition

A_Data byte	Parameter Name	Cvt	Byte Value	Mnemonic
#1	initiateActivation FID	M	0x1C	IA

2.1.1.73 Positive Response Message

Table 80 — Positive response message definition

A_Data byte	Parameter Name	Cvt	Byte Value	Mnemonic
#1	initiateActivation Response FID	M	0x9C	IAPR
	activationTime[]=[			AT_
#2	Byte#1 (MSB)	M	0x00 – 0xFF	B1
#3	Byte#2]	M	0x00 – 0xFF	B2

Table 81 — Request message parameter definition

Definition
activationTime This represents the worst case amount of time required starting when the ECU receives a function ID 0x1C (initiateActivation) request, continuing until it performs all actions necessary to activate the newly downloaded software, and ending when the new software is actively executing and able to positively respond with the part number requests. Until new software is up and running, the server shall not acknowledge or process any new OTA requests. Data type: Unsigned Numeric Resolution: 1, Offset: 0, Units: seconds Range: 0 – 65535 sec

2.1.1.74 Supported negative response codes

Table 82 — Supported negative response codes

NRC	Description	Mnemonic
0x13	**incorrectMessageLengthOrInvalidFormat** This NRC shall be sent if the length of the request message is invalid.	IMLOIF
0x22	**conditionsNotCorrect** This NRC indicates that the requested action will not be taken because the server prerequisite conditions are not (e.g., Vehicle is not in safe state)	CNC
0xF8	**IncompatibleSoftware** This NRC indicates that the server detected incompatibility within different SW components (Say for example, Strategy and Cal-Configuration are not compatible). Note: This compatible check is in addition to Swash check. This is module specific requirement. If Target ECU (say for example, ABS) have specific requirement to perform this additional check, then, this NRC shall be supported.	IS
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

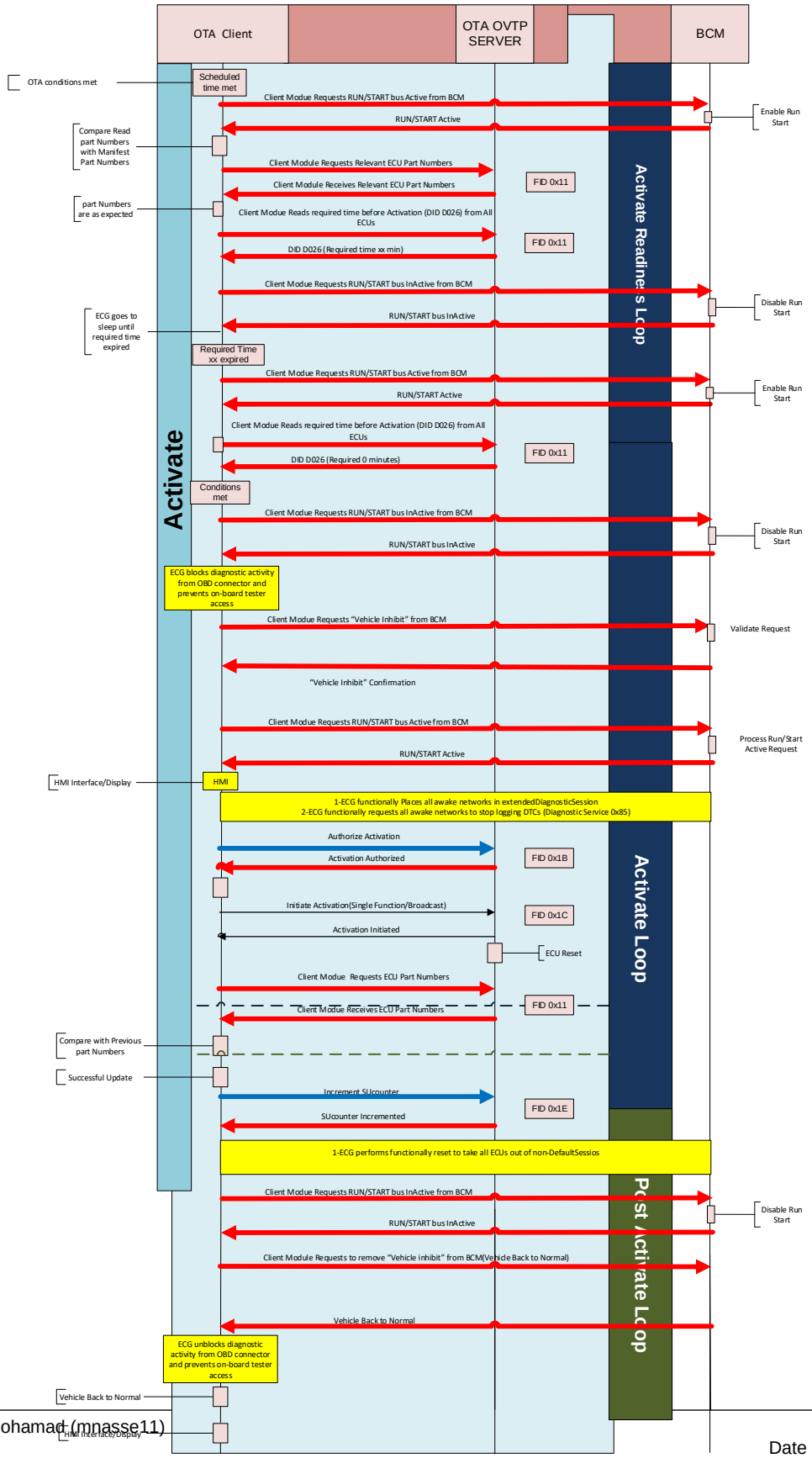


Figure 8: Example Activation Flowchart

REQ-308110/F-initiateRollBack (0x1D) Function

2.1.1.75 Function Description

The initiateRollBack function provides the means for the client to authorize and initiate rollback from a previous software activation. The initiateRollBack function handles the complete rollback process from the current memory to the target rollback memory. The rollback process validates all logical blocks of the target rollback memory. Upon accepting this request via a positive response, the server shall perform the rollback to the inactive memory and perform any necessary actions (e.g., reset) for the currently inactive application to become active.

initiateRollback is a category 1 signed OTA request (see REQ-333407). If the request is deemed valid as described in REQ-333407, the server shall validate that every single verificationStructureAddress it supports (excluding those corresponding to differential areas) is included in the request with no additional verificationStructureAddresses being present.

The SWash calculation shall be performed over the target rollback memory and shall use the order of the verification structure addresses as specified in Appendix D. The SWash is used as a mechanism to verify the complete set of software that will be rolled back to matches what is expected. The calculated SWash over what will become the active application if an initiateRollback is received must match the SWash in the request in order for this request to be accepted by the server.

2.1.1.76 OVTP Header Information

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

Table 83 — initiateRollBack Header Info

Message	Cntr	CryptoType	SSN
Request	0	0	1
Response	0	0	1

2.1.1.77 Request Message

Table 84 — Request message definition

A_Data byte	Parameter Name	Cvt	Byte Value	Mnemonic
#1	initiateRollBack FID	M	0x1D	IRB
	FESN[] = [			FESN
#2	FESN#1 (MSB)	M	0x00 – 0xFF	B1
:	:	:	:	:
#9	FESN#8]	M	0x00 – 0xFF	B8
	SUCounter[] = [			SUC_
#10	Byte#1 (MSB)	M	0x00 – 0xFF	B1
#11	Byte#2	M	0x00 – 0xFF	B2
#12	Byte#3	M	0x00 – 0xFF	B3
#13	Byte#4]	M	0x00 – 0xFF	B4
#14	triggerType	M	0x00 – 0xFF	TT

A_Data byte	Parameter Name	Cvt	Byte Value	Mnemonic
	verificationStructureAddress[] = [			VSA_
#15	Byte#1 (MSB)	M	0x00 – 0xFF	B1
#16	Byte#2	M	0x00 – 0xFF	B2
#17	Byte#3	M	0x00 – 0xFF	B3
#18	Byte#4]	M	0x00 – 0xFF	B4
:	:	:	:	:
#n-291	verificationStructureAddress[] = [			VSA_
#n-290	Byte#1 (MSB)	M	0x00 – 0xFF	B1
#n-289	Byte#2	M	0x00 – 0xFF	B2
#n-288	Byte#3	M	0x00 – 0xFF	B3
	Byte#4]	M	0x00 – 0xFF	B4
	SWash[] = [			SWASH
#n-287	SWash#1 (MSB)	M	0x00 – 0xFF	B1
:	:	:	:	:
#n-256	SWash#32]	M	0x00 – 0xFF	B32
	rollbackSignature[] = [			RBSIG
#n-255	Byte#1 (MSB)	M	0x00 – 0xFF	B1
:	:	:	:	:
:	:	:	:	:
#n	Byte#256	M	0x00 – 0xFF	B256

Table 85 — Request message data-parameter definition

Definition
FESN The ECU FORD Serial Number is a unique numeric identifier for the module being flashed
SUCounter This parameter is a 4byte counter that is transmitted between the cloud and the module to ensure that any authorized OTA request is fresh.
triggerType Trigger Type defines what action the ECU will take when receiving FID 0x1D (InitiateRollBack). States include: 0x00 : Immediate 0x01 – 0xFF: Reserved by document
verificationStructureAddress The verificationStructureAddress contains the beginning address of a verification structure when the ECU implements software signing.
SWash The Software Hash is a hashing function of all the data partitions being updated by OTA (Refer to Appendix D).
rollbackSignature This parameter is the secure command signature that ensures the request was authorized by Ford backend systems.

2.1.1.78 Positive Response Message

Table 86 — Positive response message definition

A_Data byte	Parameter Name	Cvt	Byte Value	Mnemonic
#1	initiateRollBack Response FID	M	0x9D	IRBPR
	rollBackTime[]=[			AT_
#2	Byte#1 (MSB)	M	0x00 – 0xFF	B1
#3	Byte#2]	M	0x00 – 0xFF	B2

Table 87 — Request message parameter definition

Definition
rollbackTime This represents the worst case amount of time required starting when the ECU receives a function ID 0x1D (initiateRollBack) request, continuing until it performs all actions necessary to rollback to the previous software, and ending when the new software is actively executing and able to positively respond with the part number requests. Until new software is up and running, the server shall not acknowledge or process any new OTA requests. Data type: Unsigned Numeric Resolution: 1, Offset: 0, Units: seconds Range: 0 – 65535 sec

2.1.1.79 Supported negative response codes

Table 88 — Supported negative response codes

NRC	Description	Mnemonic
0x13	**incorrectMessageLengthOrInvalidFormat** This NRC shall be sent if the length of the request message is invalid.	IMLOIF
0x15	**endToEndSignatureInvalid** This NRC indicates that an embedded end to end signature in the function was deemed invalid by the server.	ETESI
0x16	**FESNInvalid** This NRC indicates that an embedded FORD electronic serial number in the function was deemed invalid by the server.	FESNI
0x17	**softwareUpdateCounterInvalid** This NRC indicates that an embedded software update counter in the function was deemed invalid by the server.	SUCI
0x22	**conditionsNotCorrect** This NRC indicates that the requested action will not be taken because the server prerequisite conditions are not met (e.g., Vehicle is not in safe state)	CNC
0x31	**requestOutOfRange** This NRC shall be sent if triggerType or verificationStructureAddress is not valid or supported by the server.	ROOR
0x72	**generalProgrammingFailure** This NRC indicates that the server detected an error when erasing or programming a memory location in the device.	PF
0x79	**validationFailed** This NRC shall be sent if the provided SWash does not match the ECU's calculated SWash.	VF

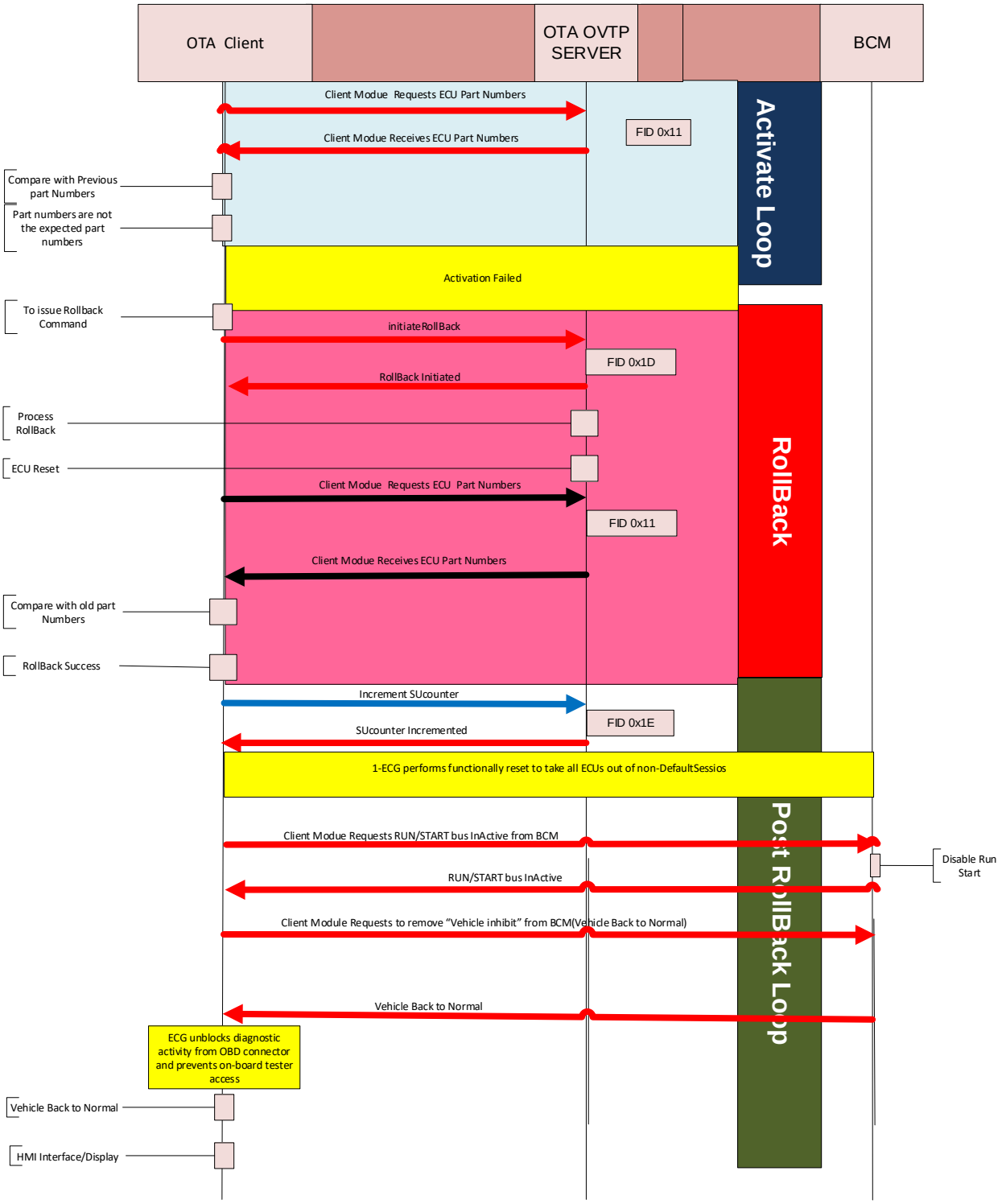


Figure 9: Example RollBack Flowchart

REQ-308111/C-initiateForceSyncCounter (0x1E) Function

2.1.1.80 Function Description

The initiateForceSyncCounter function provides the means for the client to update the ECU's stored SUCounter for the purpose of invalidating authorization requests with a lower SUCounter value.

initiateForceSyncCounter is a category 1 signed OTA request (see REQ-333407). If the request is deemed valid as described in REQ-333407, the server shall update the SUCounter and provide a positive response only if all of the following hold:

1) The received SUCounter is higher than the stored SUCounter. The one exception to this is if the stored SUCounter is greater 0xFFFFFF00 (i.e., in this scenario a lower SUCounter can be accepted).
2) The received SUCounter is not equal to 0xFFFFFFFF

2.1.1.81 OVTP Header Information

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

Table 89 — initiateForceSyncCounter Header Info

Message	Cntr	CryptoType	SSN
Request	0	0	1
Response	0	0	1

2.1.1.82 Request Message

Table 90 — Request message definition

	Parameter Name	Cvt	Byte Value	Mnemonic
#1	initiateForceSyncCounter FID	M	0x1E	IFSC
#2 : #9	FESN[] = [FESN#1 (MSB) : FESN#8]	 M : M	 0x00 – 0xFF : 0x00 – 0xFF	FESN B1 : B8
#10 : #13	SUcounter[] = [Byte#1 (MSB) : Byte#4]	 M : M	 0x00 – 0xFF : 0x00 – 0xFF	SUC B1 : B4
#22 : #270	FSCSignature[] = [Byte#1 (MSB) : Byte#256	 M : M	 0x00 – 0xFF : 0x00 – 0xFF	FSCS B1 : B256

Table 91 — Request message data-parameter definition

Definition
FESN The ECU FORD Serial Number is a unique numeric identifier for the module being flashed
SUCounter

Definition
This parameter is a 4byte counter that is transmitted between the cloud and the module to ensure that any authorized OTA function is fresh.
FSCSignature This parameter is the secure command signature that ensures the request was authorized by Ford backend systems.

2.1.1.83 Positive Response Message

Table 92 — Positive response message definition

A_Data byte	Parameter Name	Cvt	Byte Value	Mnemonic
#1	initiateForceSyncCounter Response FID	M	0x9E	IFSCRP

2.1.1.84 Supported negative response codes

Table 93 — Supported negative response codes

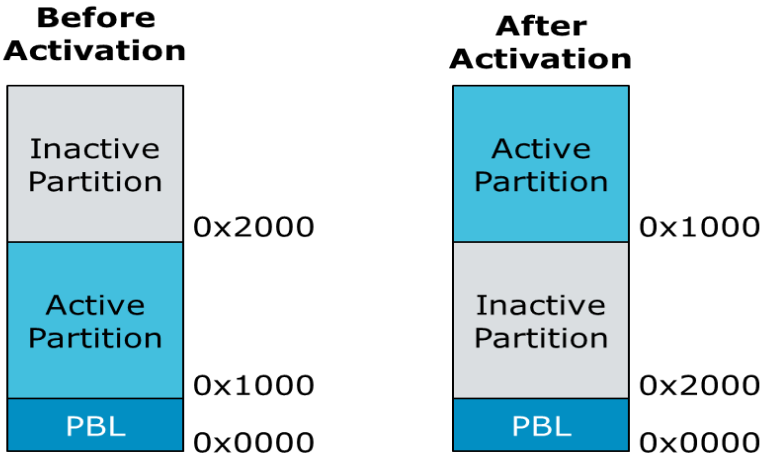
NRC	Description	Mnemonic
0x13	**incorrectMessageLengthOrInvalidFormat** This NRC shall be sent if the length of the request message is invalid.	IMLOIF
0x15	**endToEndSignatureInvalid** This NRC indicates that an embedded end to end signature in the function was deemed invalid by the server.	ETESI
0x16	**FESNInvalid** This NRC indicates that an embedded FORD electronic serial number in the function was deemed invalid by the server.	FESNI
0x17	**softwareUpdateCounterInvalid** This NRC indicates that an embedded software update counter in the function was deemed invalid by the server.	SUCI

3 Appendix A

The following are high level example ECU architectures which could be used to meet the Ford OTA requirements. These examples are not intended to be all inclusive or limiting.

3.1 OTA Architecture Type 1 – Hardware Facilitated Address Remapping

With this approach, activation of a partition involves remapping the active and inactive memory address spaces. This is normally achieved in hardware through the writing of a register or user configuration block.



High Level Requirements:

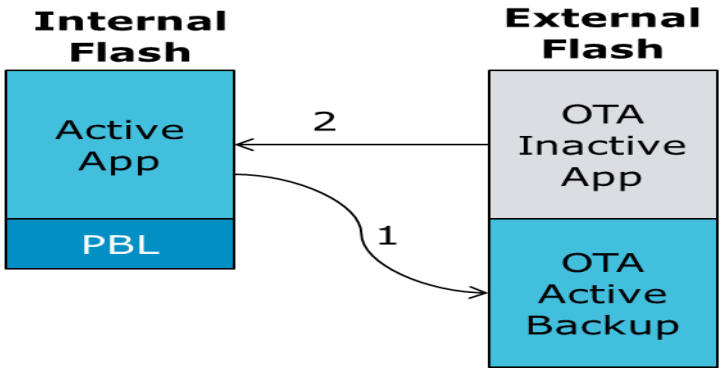
- Hardware assisted memory remapping
- 2x internal flash to support storage of both A & B memory
- Read-while-write capability to internal flash

General flow comments

In initiateActivation, the ECU shall execute the necessary actions (example: writes register/UCB) to perform the memory remapping and resets. This assumes the SWash calculation provided in the authorizeActivation request already verified is still valid .

3.2 OTA Architecture Type 2 –Memory Caching Option 1

With this approach, the new software is downloaded in the background into an allocated external memory area. Prior to activation of the new software, the currently active application is backed up into external memory and the new software is then copied into the active internal memory by the bootloader.



High Level Requirements:

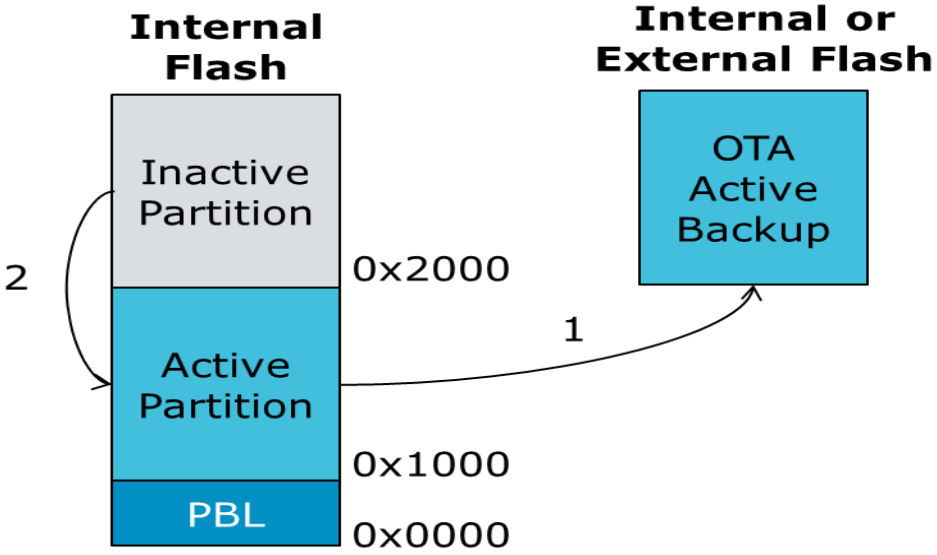
- 2x external flash to support storage of both A & B memory

General flow comments

1. Prepare for activation – The ECU will erase external flash and copy active application into external flash (in case it is needed for rollback). In case of failure during activation, the ECU shall be able to rollback using the OTA Active Backup copy automatically without the need for rollBack FID.
2. Perform activation and reset – The ECU will erase internal flash and copy the new software from external flash into internal flash. This assumes the SWash calculations match both in the OTA Inactive Map prior to beginning the erase and copy, and also the SWash calculations match in the Active App after copying prior to activation.

3.3 OTA Architecture Type 3 – Memory Caching Option 2

With this approach, the new software is downloaded in the background into an allocated internal memory area. Prior to activation of the new software, the currently active application is backed up into a dedicated backup location in either internal or external memory and the new software is then copied from the inactive internal partition to the active internal partition by the bootloader. The position independent code issue is addressed since the software is always running from the same memory address.



High Level Requirements

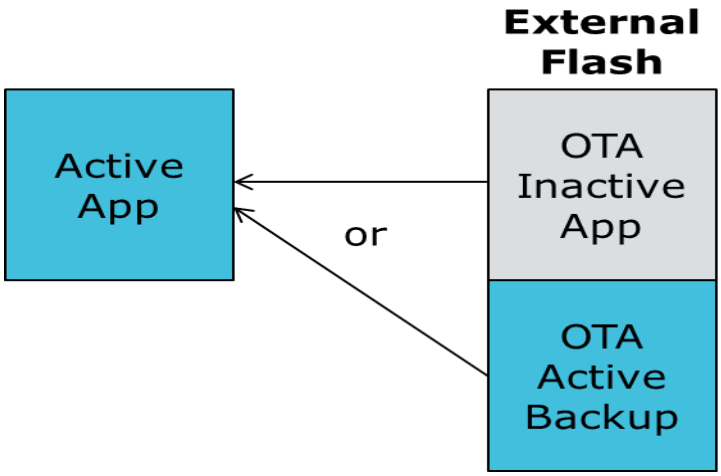
- 3x memory to support storage of both A & B memory along with backup

- Read-while-write capability to internal flash
- down time required to copy the internal memory to internal

General flow comments
1. Prepare for activation – The ECU will erase the backup memory area and copy active application into this area in case of rollback
2. Perform activation and reset – The ECU will erase internal flash and copy the new software from the inactive partition of internal flash to the active partition of internal flash. This assumes the SWash calculations match both in the Inactive Partition prior to beginning the erase and copy.

3.4 OTA Architecture Type 4 – Execute from RAM

With this approach, the software is compiled to run from a fixed location in RAM. On startup, a lookup table is used to determine which partition is copied into RAM. The position independent code issue is addressed since the software is always running from the same memory address (in RAM).



High Level Requirements
- 2x memory to support storage of both A & B memory along with backup
- Sufficient RAM to execute the application
- on microcontrollers with sufficient RAM, but often only a viable option for system on a chip configurations

General flow comments
1. Perform activation and reset – The ECU will update the lookup table and resets. This assumes the SWash calculation provided in the activation request matches prior to updating lookup table and resetting.

4 Appendix B

4.1 Reference Documents

Unless otherwise stated, the latest version of each reference document shall apply.

Table 94 — Specifications Reference

Reference #	Document Title
1	On Vehicle Telematics Protocol Specification
2	Command Signing Security Specification
3	Software Signing Specification Version
4	Data Encryption and Compression Specification
5	FESN Generation Specification
6	Software Download Specification

5 Appendix C

5.1 OTA OVTP DIDs

This section contains additional details regarding certain OTA DIDs required to be supported by the target ECU as defined in REQ-308098. Refer to Ford's Global Master Reference Database (GMRDB) for full definitions.

REQ-333419/B-DID D022

Name: In Progress OTA Download Address
Type: Packeted
Size: 5 bytes
Parameter #1:
Name: OTA Initiate Download Function in Progress
Type: State Encoded
Size: 1 byte
Values:

- 0x00 = False
- 0x01 = True

Parameter #2:
Name: OTA Last Address Successfully Written
Type: Hex
Size: 4 bytes

Purpose: If an initiateDownload has been accepted and not all data has been transferred, the server shall always report a value of 0x01 for parameter #1 and the last written address for parameter #2. Parameter #1 shall only be set back to 0x00 when one of the following occurs:

- All requested bytes from that initiateDownload have been received via the transferData requests (no completeDownload is needed to set this back to 0x00).
- A prepareActivation, eraseMemory or diffUpdate has been positively responded to

The intent of this DID is to ensure the client can synchronize with the last memory address actually written by the server if an initiateDownload function was started but not all data was sent in order for the client to correctly resume the transfer of data.

After acceptance of a new initiateDownload, DID 0xD022 shall report the initiateDownload accepted address -1. In the case where the initiateDownload was accepted at address 0x00000000, then 0xD022 shall report 0xFFFFFFFF.

For example if the initiateDownload is accepted for address X, the first byte of 0xD022 shall report 0x01 and the second byte shall report X-1.

REQ-333420/B-DID D026

Name: OTA Activation Preconditions
Type: Packeted
Size: 2
Parameter #1

Name: OTA Activation Precondition Status
Type: State Encoded
Size: 1 byte
Value: 0x00 = All preconditions met
0x01 = One or more OTA preconditions not met

Parameter #2
Name: Ignition Off Time Required Prior to Accepting InitateActivation of New OTA Software
Type: Unsigned Numeric
Size: 1 byte
Resolution: 1
Offset: 0
Units: minutes

Purpose: The intent of this DID is to provide a status to the client on whether the ECU could swap to new software if requested to now. It includes the worst case estimate of how much time is needed if the ignition were switched off now before it could affirmatively respond to an initiateActivation request (assuming valid software is present in the inactive partition and an authorizeActivation was received). For most ECUs, it is anticipated they will always report a value of 0 minutes to signify they do not have a time based precondition which automatically clears (e.g., uninterruptible after-run). However, some ECUs may perform an after-run treatment such as a cool-down which should not be interrupted in order to prevent damage. It also includes any ECU specific preconditions that may not be currently met. If doing a coordinated update, the client needs to help ensure all conditions are met prior to sending the initiateActivation request to all ECUs.

When Parameter #1 “OTA Activation Precondition Status” is reporting a value of 0x00, this requires that Parameter #2 is reporting a value of 0x00 (0 minutes) and furthermore that DID 0xD04F is reporting a value of all 0x00s.

When Parameter #1 “OTA Activation Precondition Status” is reporting a non zero value, this requires that DID 0xD04F is reporting at least one bit set to a value of 1. Furthermore, if no preconditions which are actively preventing OTA are time bound (but rather requires user interaction to remove) then Parameter #2 shall report a value of 0x00 (0 minutes). If one or more preconditions which are actively preventing OTA are time bound (e.g., after-run active), then Parameter #2 shall report the worst case time.

REQ-333421/A-DID D029

Name: OTA Support Level
- Reported Information
  - OTA Spec Version (e.g., 006)
  - OTA Functionality(DIDs only or full)
- Main Usage
  - Allows OTA client to verify OTA spec version in case it needs to adopt to differences

REQ-333422/A-DID D02B

Name: OTA Software Update Counter
- Reported Information
  - 4 byte counter value
- Main Usage
  - Allows OTA client to verify ECU’s value, especially for bench conformance testing

REQ-358790/A-DID D031

Name: Vehicle Conditions Preventing OTA Activation or OTA SWDL of New Software
Type: Bitmapped
Size: 4 bytes
All bitmapped parameters are defined per GMRDB

Purpose: This is a standard DID that is only implemented if an ECU has additional preconditions above and beyond DID $D04F that must be checked in order to prevent the OTA programming of itself.

This DID will only be read from ECUs which are actually undergoing an OTA update and not from all ECUs simply placed into the programmingSession using diagnostics for OTA. When this DID reports a non-zero value it will prevent OTA activation using OTA over OVTP protocol but will not prevent accepting a request to enter programmingSession due to an OTA event. In other words, a non-zero value for DID $D031 (when DID $D04F is reporting all 0x00s) indicates that the ECU can go into programmingSession due to an OTA event but does not want to be programmed itself. The onus is on the OTA client not to program the ECU in this case. A simple example is a window module that can go into programmingSession when the windows are down (so as not to prevent a SWDL OTA of another ECU), but does not want to be programmed itself in case the OTA events leaves the ECU non-functional.

REQ-333423/B-DID D03B

Name: OTA Debug Information
Type: Packeted
Size: 24 bytes
Parameter#1:
Name: Most Recent OTA FID Processed
Type: Hex
Size: 1 byte
Parameter#2:
Name: Most Recent OTA FID Response Type
Type: State Encoded
Size: 1 byte
Values:
- 0x00 = Positive response
- 0x01 – 0xFF = Decoded according to NRC table in reference [1]

Parameter#3
Name: Most Recent OTA FID Debug Info
Type: hex
Size: 4 bytes
Other: Refer to Table 91 for decoding information

Note: The 6 bytes associated with parameter #1 - #3 represent information for the most recently processed OTA FID. These 6 bytes repeat three times with parameters #4 - #6 representing the same information for the 2nd most recently processed OTA FID. Similarly, parameters #10 - #12 represent the same information for the 4th most recently processed OTA FID. All 24 bytes shall be initialized to 0x00 on new ECUs.

Only processed OTA FIDs 0x12, 0x13, 0x14, 0x15, 0x16, 0x17, 0x18, 0x19, 0x1A, 0x1B, 0x1C, 0x1D, 0x1E shall be stored in this DID. In other words, a received OTA FID of 0x11 or 0x01 shall have no effect on the data reported by this DID.

Table 91 — DID D03B Codes

FID		Response Type	4 byte Debug implementation/definition
authorizeEraseMemory	0x12	00 - FF	Echo of the SUcounter from the request message

initiateDownload	0x15	00 - FF	Echo of the memoryAddress from the request message
transferData	0x16	00	4 byte address of the last byte successfully written. Note that this shall correspond with the current value in $D022 at this point.
transferData	0x16	01-FE	Most significant byte = echo of blockSequenceCounter from requestLeast 3 significant bytes = least 3 significant bytes at which the request would have started to write if positively processed.
transferData	0x16	FF	4 byte address of the last byte successfully written.
completeDownload	0x17	00	4 byte address of the last byte successfully written. Note that this shall correspond with the current value in $D022 at this point.
completeDownload	0x17	01-FF	All $00s
validateLogicalBlock	0x19	00 - FF	Echo of the verificationStructureAddress from the request message
diffUpdate	0x18	00	Echo of the verificationStructureAddress from the request message
diffUpdate*	0x18	01 - FE	Most signficant byte* = diff engine supplier specific error codes Least 3 significant bytes = echo of the least 3 significant bytes of the verificationStructureAddress from request
diffUpdate*	0x18	FF	Most signficant byte* = diff engine supplier specific context status code Least 3 significant bytes = echo of the least 3 significant bytes of the verificationStructureAddress from request
prepareActivation	0x1A	00 - FF	Echo of the 4 least signficant bytes of the SWASH from the request message
authorizeActivation	0x1B	00 - FF	Echo of the SUcounter from the request message
initiateRollback	0x1D	00 - FF	Echo of the SUcounter from the request message
initiateForceSyncCounter	0x1E	00 - FF	Echo of the SUcounter from the request message
initiateActivation	0x1C	00 - FF	All $00s
eraseMemory	0x13	00 - FF	Echo of the memoryAddress from the request message
authorizeDownload	0x14	00 - FF	Echo of the SUcounter from the request message

Note that if NRC is 0x13, ECU can only fill in what information it has in the 4 byte debug.
Unavailable information shall be filled with 0x00s.
For example, a FID 0x12 request with only 1 byte of A_Data will not have a SUcounter value to include.

REQ-333424/B-DID D03E

Name: In-Use OTA Command Signing Public Key Hash

- Reported Information
  - 32 bytes Hexadecimal
- Main Usage

This DID shall be used to return the SHA-256 hash of the OTA over OVTP cloud signed command public key that is currently active in the target ECU. It is expected this DID will be supported in the application code

REQ-333425/D-DID D04F

Name: OTA ProgrammingSession Entry and A/B Swap Precondition Status
Type: Bitmapped
Size: 4 bytes
Parameter #1 Name: No Preconditions Supported
0 = True, 1 = False
All other parameters names are defined per GMRDB, but with bit values of
0 = False, 1 = True

Purpose: The intent of this DID is to provide specific feedback status to the OTA client as to why an ECU is not able to transition into programmingSession (using normal diagnostic protocol) for an OTA event or why an ECU is not able to perform an A/B activation (using the OTA over OVTP protocol).

Parameter #1 is supported only if no other parameters within the DID are supported. If parameter #1 is supported, the DID shall always report a value of all 0x00s and DID 0xD026 shall always report a value of all 0x00s.

If any bit is reporting a value of 1 in this DID (i.e., the DID is reporting a non-zero value), the ECU shall not accept a request for an A/B activation swap and must not accept a request to enter programmingSession due to an OTA event.

REQ-383103/B-DID D039

Name: OTA Partition Status
Type: Bitmapped
Size: 3 bytes

Byte 1, Bit7: ECU RollBack Possible
Byte 1, Bit6 – Bit4: Undefined
Byte 1, Bit3: Partition 0 – Rollback Possible
Byte 1, Bit2: Partition 0 – Active Backup Location
Byte 1, Bit1: Partition 0 – Inactive
Byte 1, Bit0: Partition 0 – Active
Byte 2, Bit7 – Bit4: Undefined
Byte 2, Bit3: Partition 1 – Rollback Possible
Byte 2, Bit2: Partition 1 – Active Backup Location
Byte 2, Bit1: Partition 1 – Inactive
Byte 2, Bit0: Partition 1 – Active
Byte 3, Bit7 – Bit4: Undefined
Byte 3, Bit3: Partition 2 – Rollback Possible
Byte 3, Bit2: Partition 2 – Active Backup Location
Byte 3, Bit1: Partition 2 – Inactive
Byte 3, Bit0: Partition 2 – Active

Purpose: The intent of this DID is to provide information on the ECU partition and status. A/B ECUs would not support bits in Byte 3 (as they don't have a 3rd partition). Therefore, a non-zero value in Byte 3 then always indicates this is an "A/B/A architecture", whereas a 0x00 value in byte 3 always indicates it is an "A/B architecture".

The ECU shall set Byte 1 bit 7 to “1” whenever rollback is possible regardless of the ECU architecture and regardless of partition.

Note that Active Backup Location and Inactive do not imply that the partition contains valid code. However, Rollback Possible is set on a swap and does indicate that the ECU should be able to accept a rollback request. This bit is cleared when the partition is erased or modified.

6 Appendix D

REQ-333426/C-SWASH Details

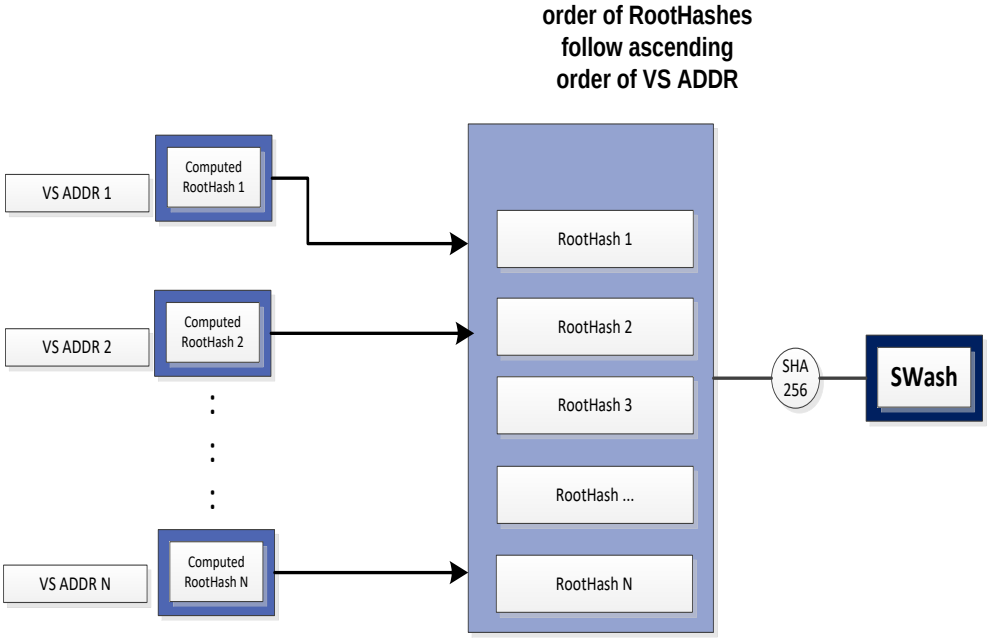


Figure 10: SWash Calculation

1. Computed RootHash shall be calculated based on Ref 3.
2. Per Target ECU, one or more VBF shall be present.
3. Per VBF, one or more Verification structure present. So, one or more VS ADDR and Computed RootHash shall be present.
4. Diff logical block VSA and Computed RootHash is not part of SWash calculation
5. Big Endian used for storing and calculation.

Note: Irrespective of VBF files are created for a particular Target ECU for a particular SW update, Target ECU shall operate on number of VSAs predefined during design time.

For example:
Target ECU with 3 VSA and associated RootHashes

VSA	-	RootHash
0x801FFF00	-	0xCF6822974AA52F6E596B81EB366529AA19B270CB6F615F85BA11FBC9362218D6
0x803FFF00	-	0x7648A086A5FA30B4F62FF44CADD7B90D3F70952024DFCD9A50D7AE44846F17BB
0x805FFF00	-	0xB4B55A0087DFCB59F99CE42E4C92E9EF111421DA2ED6FA3395996B872D4990B9

SWash = Sha256
(CF6822974AA52F6E596B81EB366529AA19B270CB6F615F85BA11FBC9362218D67648A086A5FA30B4F62FF44CADD7B90D3F70952024DFCD9A50D7AE44846F17BBB4B55A0087DFCB59F99CE42E4C92E9EF111421DA2ED6FA3395996B872D4990B9)

SWash = 0xEC43A131154FA4B635A420D7D5A634B300F89529272EE765A79CECF05D36A54B

Target ECU with 1 VSA and associated RootHash

VSA	-	RootHash
0x801FFF00	-	0xCF6822974AA52F6E596B81EB366529AA19B270CB6F615F85BA11FBC9362218D6

SWash = Sha256 (CF6822974AA52F6E596B81EB366529AA19B270CB6F615F85BA11FBC9362218D6)
SWash = 0x30EE1F8D1CBBF3A7FB8CD33A73F68CDF42B779B8B728E5D716D8CAC15D532633

Notes
The SWASH calculation may be used at various times for the OTA process. For ABA (extended use case), the safety related SWASH calculation in the bootloader should be calculated from scratch(over the entire logical blocks) without using previously stored intermediate hashes.