

Software Download Specification

Diagnostic Communications

# Revision History

Release: 008 Date: 2022-03-28 Author: John Cardillo / Jason Miller

|  |  |
| --- | --- |
| Section | Change Description |
| 3.6.9, 3.6.10 | Clarified behavior for A/B implementations which do not have a separate active backup. |
| Table 2.1, Table 2.2 | Moved routine 021CH support from PBL to SBL |
| 3.6.10 | Add prerequisite condition for routine 021CH that it was not previously successfully executed during programmingSession |
| 3.6.1 | Replaced 0306H with 0307H to support newly formatted routine with enhanced error handling |
| Table 3.22 | Fixed typo in table to change SWDL version from 7 to 8. |
| Several | Corrected bad cross-references. |
| Table 1.1 References | Updated VDOC hyperlinks for latest test specifications and added new hyperlink for SWDL Functional Test Spec |
| All | Updated case of many headings and various formatting updates |
| All | Updated usage of sub-node and sub-network to be used consistently |
| 3.3 | Added requirement that fixed bytes are randomly generated independently for each level. |
| 1.7 | Added definitions for A/B and A/B/A |
| Table | Fixed type of routine number in Table description |
| 3.3 | Updated requirements to explicitly call out all diagnostic functionality which can never be executed without security access performed. |
| Figure 2.1 | Updated values to be more realistic (e.g., PBL from 16k to 64k) |
| Table 3.22 | Updated Table to reflect SWDL specification version reported is 8. |
| 3.6.4 | Added text to clarify memory areas routine FF01H must support. |
| Figure 6.1 | Updated arrow to show flow going from Sleep mode to Normal mode and not directly to bootloader |
| 10.1 | Updated explicit gateway text to align with [Ford GGDS]. |
| 4.4, 3.6.7, 3.6.10 | Modified the routines and details on how updates of the inactive partition and rollback to the previously active software are performed in order to robustly support all memory architectures. |
| Many | Added official routine numbers of 021BH for “Set OTA Update State” and 021CH for “Replace Active memory Partition” |
| Table 7.2 | Added additional P4 time for routine 021AH and 021CH. |
| Table 1.1 | Changed GMRDB definition to “Global Main Reference Database” |
| 11.1 | Added D028H, D03FH to list of DIDs that are not required for ECUs that do not implement a Boot Loader. |
| 3.6 | Removed routineIdentifier 0306H. This routine is not used in the programming session and adds unnecessary complexity. |
| Many | Added updates to support UDS programming details of inactive memory partitions |
| 3.3 | Added fixed byte security restriction to not allow byte values that repeat 3 or more times. |
| 3.3 | Deleted text describing validity of a seed as it conflicted with normative requirements from ISO 14229-1. |
| Annex D | Removed details from outdated Appendix D related to conversion of a programmable part to a Masked ROM |
| 3.1, 3.6.3, 6, 7.1, 7.2, 7.3, 11.1 | Updated wording to replace P2CAN, P2\*CAN, P3CAN, and P4CAN with P2, P2\*, P3, and P4 as they apply independent of lower OSI layers. |
| 8.1.1.1 | Modified text of reserved CAN identifier ranges for diagnostics to reference Table 8.1 to avoid conflicts. |
| 1.9.1, 11.1, 11.2 | Added new ECU programming variants for ECUs programmable via a replaceable application and Autosar Adaptive ECUs. |
| Annex E | Added clarification that the software identification DIDs only apply to ECUs that release their software through WERS. |
|  | Copyright and editorial changes |
| 1.6, 1.12, 8, 9 | Added support for Diagnostics over Internet Protocols. |

Release: 007.2 Date: 2021-03-22 Author: John Cardillo

|  |  |
| --- | --- |
| Section | Change Description |
| 3.6.8 | Clarified the expected behavior of routine 0213H (Activate Inactive Partition). The routine is expected to fully perform any required memory swaps before returning the positive response. |
| 2.1 | Removed the requirement for the primary bootloader to enable I/O that is required to power other ECUs that support software download. Power management requirements have been moved to [FORD\_VPM]. |
| 3.3 | Added Security Seed and Shared Secret requirements aligned with [Ford GGDS]. |
| All | Reformatted document to new corporate standards. |

Release: 007.1 Date: 2020-08-08 Author: John Cardillo

|  |  |
| --- | --- |
| Section | Change Description |
| Table 2.1  3.6.7 | Generalized the use of the term A/B swap to clarify that requirements apply to all forms of swaps involving multiple memory partitions. |
| 3.6.1 | Added requirement for explicit gateways that support routine 0300H to also support routine 0306H. This change simplifies and speeds up the collection of part numbers from an explicit gateway’s sub-nodes. |
| Table 2.1 | Clarified footnote MD5 to note that FESN (DID F17F H) is only mandatory if specified in the ECU’s requirements. |
| 8.1.3.1  8.1.3.2 | With approval from Ford Network Communications, if an ECU has a separation time STmin greater than 00H and is behind a gateway that cannot preserve the reported STmin, then the ECU may mitigate this issue by specifying a BlockSize of 01H to force a flow control frame between each consecutive frame. |
| Table 7.2 | Added Note 5 to Service 31H (routineControl) with routine FF01 H (check ProgrammingDependencies) to account for the size of the program being validated. |

Release: 007 Date: 2018-10-30 Author: Bill Waldeck

|  |  |
| --- | --- |
| Section | Change Description |
| All | Release with all changes from above |

Release: 006.4 Date: 2018-10-30 Author: Bill Waldeck

|  |  |
| --- | --- |
| Section | Change Description |
| 2.2.1, 3,5,4 | Added requirement that FESN (DID F17F) a “write-once” DID |

Release: 006.3 Date: 2018-08-21 Author: Bill Waldeck

|  |  |
| --- | --- |
| Section | Change Description |
| Table 4, 3.4.9, 3.6.7 | ERRATA 5 Modified: replaced Routine 0212H with DID D03FH |
| Table 73 | ERRATA 17: Added 640H – 67F H to the UDST range to increase the available CAN IDs |
| Table 4, 3.4.X(9) | ERRATA 18: For DID F18C added format requirement if the ECU must meet traceability requirements. Also added FESN (DID F17F) as mandatory / conditional |

Release: 006 Date: 2013-05-02 Author: Jason Miller

|  |  |
| --- | --- |
| Section | Change Description |
| 1.4 | Removed reference to NetCom eRoom. |
| Revision history | ERRATA 1: The official release incorrectly listed “006” as the Previous Version and “005.5” as the Current Version. These are reversed. |
| 3.3 | ERRATA 2: Removed sentence that states the security algorithm can be obtained from Ford on request. Even if the bootloader is not purchased from Ford the algorithm still must be purchased from Vector. Added security requirement for Routine 0x0213H |
| Table 69 | ERRATA 3: Added Cal-Config #7 DID F10E |
| Table 4, 3.4.8 | ERRATA 4: Added DID D028H for Application Signature Validation Status |
| Table 4, 3.6.7 | ERRATA 5: Added Routine 0212H Calculate Public Key Hash |
| Table 4, 3.6.8 | ERRATA 6: Added Routine 0210H Use Inactive Memory Partition |
| Table 4, 3.6.9 | ERRATA 7: Added Routine 0213H Activate Inactive Memory Partition |
| Table 73 | ERRATA 8: Increase 11-bit CAN ID range for USDT to 6C0H –7FFH |
| 7.2.2.3 | ERRATA 9: Allow exceptions to Explicit Gateway State Access Not Active when approved by Ford Core Network Communications |
| Table 1 | ERRATA 10: Change the version and date of [ISO 15765-2] to the April 2016 version of [ISO 15765-2] which includes support of CAN FD. |
| 1.5, 1.6 | ERRATA 11: Add CAN FD related abbreviations and definitions |
| 8.3 | ERRATA 12: Add new section for CLASSICAL CAN and CAN FD requirements |
| 7.3.7 | ERRATA 13: Add new section limiting the maximum message size to 4095 bytes. |
| 7.3.8 | ERRATA 14: Clarified use of the new [ISO 15765-2] parameter, TX\_DL |
| 7.2.2 | ERRATA 15: Clarified that the CAN IDs used for CLASSICAL CAN and CAN FD shall be the same. |
| 8.1, 8.2 | ERRATA 16: Clarified CLASSICAL CAN and CAN FD frames sizes and padding |

Release: 005.5 Date: 2013-04-22 Author: Jason Miller

|  |  |
| --- | --- |
| Section | Change Description |
| Table 1 | Updated to latest published version of [ISO 14229-1] and [ISO 14229-1]. |
| 4.1 | Clarified that part of the recommended programming sequence is for the tool to send all functional requests on all public networks at the DLC. |
| 3.6 | Updated request for writing security bytes for security level 01H to reflect the more generic GMRDB definition for routine 0305H where the security level to write is included in the request. |
| 7.2.2.4 | Clarified that the sizing of the first-in first-out queues for a frame based explicit gateway are the responsibility of the diagnostic feature owner. |

Release: 005.4 Date: 2013-02-06 Author: Jason Miller

|  |  |
| --- | --- |
| Section | Change Description |
| 3.4 | Clarified that the requirement to embed the part number in the data and report via a DID is not applicable to the SBL as it is only temporarily downloaded into RAM. |

Release: 005.3 Date: 2012-10-03 Author: Jason Miller

|  |  |
| --- | --- |
| Section | Change Description |
| 7.3.6 | Increased network/transport layers timeouts to 1000ms for consistency with the application and to support more robust downloads through gateways. |
| Table 4, Table 5 | Moved support of routine 0305H only to SBL. |

Release: 005.2 Date: 2012-05-08 Author: Jason Miller

|  |  |
| --- | --- |
| Section | Change Description |
| 3.3 | Added clarification that securityAccess service is intended primarily to protect downloading of information to RAM (e.g., SBL). |
| 6.2 | Added clarification that the actual mechanism for determining if software is valid for a given hardware part is not mandated by this specification. |
| All | Miscellaneous editorial changes. |
| 4 | Added clarification that while dependencies should be kept to a minimum, any software dependencies must be agreed to between Ford design and release engineer and supplier and captured within the Ford software vault. |
| All | Clarified location of routineInfo as still in the fifth byte of positive response. The latest [ISO 14229-1] has added this field as a new parameter that occurs before the routineStatusRecord field. |

Release: 005.1 Date: 2012-01-26 Author: Jason Miller

|  |  |
| --- | --- |
| Section | Change Description |
| All | Editorial changes and clarifications |
| Change Log | Removed changes from previous version |
| 1.4 | Added references to latest ISO specifications including new specifications [ISO 14229-2], [ISO 14229-3], and removed obsoleted specifications such as ISO 15765-3 |
| 3.4.11 | Deleted section 3.4.11 (NOS Bootloader DIDs) as individual component DIDs have been replaced by a single "package" DID mandated by NOS specifications. |
| 8 | Added clarification text to state data link layer shall be compliant to reference [Ford CAN DL&PHY]. |
| Table 4 | Added allowance for routine $0304 to be supported only by SBL when explicit authorization is granted. |
| 3.3, 10.1 | Deleted requirement that service $31 is always protected by security. The standard routines that require security ($0301 and $FF00) are implicitly blocked as SBL is required. |
| All | Update all request message flows where the service supports SPRMIB to more clearly indicate this bit can be either 0 or 1. |
| 3.7, 3.10.1, 3.10.3 | Added clarification on the dataFormatIdentifier value description to better indicate this represents whether the data is compressed or encrypted as compared to what must be written into the ECU's memory. |
| All | Delete all references to Volvo's KDP part numbering system and related DIDs. |
| All | Replaced "responsible authority appointed by operating company" with appropriate text of either Ford Core NetCom or Diagnostic Feature Owner. |
| 1.5 | Added definition for Diagnostic Feature Owner. |
| 3.7.1 | Added exception to minimum value for maxNumberOfBlockLength parameter. |
| Table 4, 3.4.6, 6.1 | Added clarifications on bootloader defaultSession to clarify it is not the same as the defaultSession within the application. |
| 4.3.2 | Deleted section on how to send segmented request to not duplicate [ISO 15765-2]. |
| 2.2.1 | Added clarification that transition from programmingSession to programmingSession must reset security and an activated SBL, thereby allowing a new download into RAM. |
| 1.2 | Inserted new section for Deviations |
| All | Better clarified routine 0x0300 details are requirements are only for explicit diagnostic gateway and not relevant for CAN transparent diagnostic gateways. |
| Table 80 | Added updated allowances for routine 0x0304, service 0x37, and service 0x11 for P4CANmax. |
| 3.6.7 | Clarified NRC for a routine that requires SBL but is requested in the programmingSession of PBL. |
| 6.1.1 | Clarify request messages that allow entry to bootloader via backdoor method |
| 3.3 | Updated security access requirements to include reference to new security algorithm, reference flow chart from [ISO 14229-1], etc. |
| Fig. 4-1, Fig. 5-1 | Updated recommended programming flow chart to recommend the functional enter programmingSession is sent periodically. |
| All | Removed all PAG and non-PAG specific details |
| 3.9.3 | Added requirements for Early Acknowledgement strategy with service 36H |
| All | Deleted references to DID $F102 as it applied only to 5 fixed byte security constant |
| 3.5 | Added note to recommend suppliers have mechanism to update specific writeable identification values prior to delivery to Ford |
| 3.6.6, Table 4 | Added new optional routine to allow writing of fixed security bytes. Unlike the writeable DID approach, the routine is future proof as the fixed byte size is not pre-determined. |
| 15, 2.4 | Added Appendix E to provide overview of programming possibilities and identification options. |
| Table 1 | Added reference to Data Compression and Encryption Specification |

Release: 005 Date: 2007-06-27 Author: Jason Miller

Release: 004 Date: 2004-08-17 Author: Jason Miller

Release: 003 Date: 2004-05-14 Author: Olof Hansson

Release: 002 Date: 2003-09-15 Author: Olof Hansson

Release: 001 Date: 2002-11-27 Author: Olof Hansson

# Contents

[1 Introduction 14](#_Toc99383807)

[1.1 Purpose/Scope 14](#_Toc99383808)

[1.2 Target 14](#_Toc99383809)

[1.3 Use of This document 14](#_Toc99383810)

[1.4 Traceability 15](#_Toc99383811)

[1.5 Verification Method 15](#_Toc99383812)

[1.6 References 15](#_Toc99383813)

[1.7 Abbreviations, Acronyms and Definitions 17](#_Toc99383814)

[1.8 File Formats 18](#_Toc99383815)

[1.9 Overview 18](#_Toc99383816)

[1.9.1 Levels of Programmability 18](#_Toc99383817)

[1.10 Deviations 19](#_Toc99383818)

[1.10.1 Tester Support 19](#_Toc99383819)

[1.11 Programming Voltages 19](#_Toc99383820)

[1.12 Diagnostic Communication Requirements 20](#_Toc99383821)

[2 Two Level Bootloader 21](#_Toc99383822)

[2.1 Primary Bootloader (PBL) 21](#_Toc99383823)

[2.1.1 Primary Bootloader Services 22](#_Toc99383824)

[2.2 Secondary Bootloader (SBL) 23](#_Toc99383825)

[2.2.1 Secondary Bootloader Services 23](#_Toc99383826)

[2.3 Additional Bootloader Resources for Secondary Processor Elements 24](#_Toc99383827)

[3 Software Download Services 25](#_Toc99383828)

[3.1 DiagnosticSessionControl (10H) service 25](#_Toc99383829)

[3.1.1 Supported Negative Response Codes 25](#_Toc99383830)

[3.2 ECUReset (11H) service 26](#_Toc99383831)

[3.2.1 Supported Negative Response Codes 26](#_Toc99383832)

[3.3 SecurityAccess (27H) service 26](#_Toc99383833)

[3.3.1 Step 1: Request the Seed 27](#_Toc99383834)

[3.3.2 Step 2: Send the Key 28](#_Toc99383835)

[3.3.3 Supported Negative Response Codes 28](#_Toc99383836)

[3.4 ReadDataByIdentifier (22H) service 28](#_Toc99383837)

[3.4.1 dataIdentifier F111H (ECU Core Assembly Number) 29](#_Toc99383838)

[3.4.2 dataIdentifier F113H (ECU Delivery Assembly Number) 31](#_Toc99383839)

[3.4.3 dataIdentifier F18CH (ECU Serial Number) 31](#_Toc99383840)

[3.4.4 dataIdentifier F180H (Boot Software Identification) 32](#_Toc99383841)

[3.4.5 dataIdentifier F109H (Boot Software Version Number) 33](#_Toc99383842)

[3.4.6 dataIdentifier D100H (Active Diagnostic Session) 33](#_Toc99383843)

[3.4.7 dataIdentifier F162H (Software Download Specification Version) 34](#_Toc99383844)

[3.4.8 dataIdentifier D028H (Application Signature Validation Status) 34](#_Toc99383845)

[3.4.9 dataIdentifier D03FH (In-Use Application Signing Public Key Hash) 35](#_Toc99383846)

[3.4.10 dataIdentifier F17FH (Ford Electronic Serial Number) 35](#_Toc99383847)

[3.4.11 Supported Negative Response Codes 36](#_Toc99383848)

[3.5 WriteDataByIdentifier (2EH) service 36](#_Toc99383849)

[3.5.1 dataIdentifier F111H (ECU Core Assembly Number) 37](#_Toc99383850)

[3.5.2 dataIdentifier F113H (ECU Delivery Assembly Number) 38](#_Toc99383851)

[3.5.3 dataIdentifier F18CH (ECU Serial Number) 39](#_Toc99383852)

[3.5.4 dataIdentifier F17FH (Ford Electronic Serial Number) 40](#_Toc99383853)

[3.5.5 Supported Negative Response Codes 40](#_Toc99383854)

[3.6 RoutineControl (31H) service 41](#_Toc99383855)

[3.6.1 routineIdentifier 0300H (Gateway State Access) 41](#_Toc99383856)

[3.6.2 routineIdentifier 0301H (Activate Secondary Bootloader) 43](#_Toc99383857)

[3.6.3 routineIdentifier FF00H (eraseMemory) 44](#_Toc99383858)

[3.6.4 routineIdentifier FF01H (checkProgrammingDependencies) 45](#_Toc99383859)

[3.6.5 routineIdentifier 0304H (Check Valid Application) 46](#_Toc99383860)

[3.6.6 routineIdentifier 0305H (Update Security Bytes) 46](#_Toc99383861)

[3.6.7 routineIdentifier 021BH (Set OTA Update State) 47](#_Toc99383862)

[3.6.8 routineIdentifier 021AH (Prepare Inactive Memory) 48](#_Toc99383863)

[3.6.9 routineIdentifier 0219H (Calculate SWash) 49](#_Toc99383864)

[3.6.10 routineIdentifier 021CH (Replace Active Memory Partition) 51](#_Toc99383865)

[3.6.11 RoutineInfo 52](#_Toc99383866)

[3.6.12 Supported Negative Response Codes 53](#_Toc99383867)

[3.7 RequestDownload (34H) service 53](#_Toc99383868)

[3.7.1 RequestDownload positive response 54](#_Toc99383869)

[3.7.2 Supported Negative Response Codes 54](#_Toc99383870)

[3.8 RequestUpload (35H) service 54](#_Toc99383871)

[3.8.1 RequestUpload positive response 55](#_Toc99383872)

[3.8.2 Supported Negative Response Codes 55](#_Toc99383873)

[3.9 TransferData (36H) service 55](#_Toc99383874)

[3.9.1 TransferData positive response 56](#_Toc99383875)

[3.9.2 Supported Negative Response Codes 56](#_Toc99383876)

[3.9.3 TransferData "Early Acknowledge" Strategy (Pipelining) 56](#_Toc99383877)

[3.10 RequestTransferExit (37H) service 58](#_Toc99383878)

[3.10.1 RequestTransferExit positive response 58](#_Toc99383879)

[3.10.2 Supported Negative Response Codes 58](#_Toc99383880)

[3.10.3 Checksum algorithm 58](#_Toc99383881)

[3.11 TesterPresent (3EH) service 59](#_Toc99383882)

[3.11.1 TesterPresent positive response 59](#_Toc99383883)

[3.11.2 Supported Negative Response Codes 59](#_Toc99383884)

[3.12 Service not supported 59](#_Toc99383885)

[4 File Download Sequence 60](#_Toc99383886)

[4.1 Active Partition Download Programming Sequence - Recommended 61](#_Toc99383887)

[4.2 Active Partition Download Programming Sequence - Minimal 64](#_Toc99383888)

[4.3 Download Data Block 66](#_Toc99383889)

[4.4 Inactive Partition Download Programming Sequence 68](#_Toc99383890)

[5 File Upload Sequence 70](#_Toc99383891)

[5.1 Upload Programming Sequence 70](#_Toc99383892)

[5.2 Upload Data Block 72](#_Toc99383893)

[6 ECU Program Mode 74](#_Toc99383894)

[6.1 ECU Program Mode for Bootloader Implementations 74](#_Toc99383895)

[6.1.1 Boot Sequence for ECUs 76](#_Toc99383896)

[6.2 Validation of ECU Software 77](#_Toc99383897)

[6.3 Keep ECU programming session 77](#_Toc99383898)

[6.4 ECU Connected to Multiple Public Diagnostic Networks 78](#_Toc99383899)

[7 Application Layer 79](#_Toc99383900)

[7.1 Timing Parameters 79](#_Toc99383901)

[7.2 Diagnostic Services Response Times 79](#_Toc99383902)

[7.3 Requirements for Processing Additional Tester Requests 80](#_Toc99383903)

[8 Network Layer 82](#_Toc99383904)

[8.1 CAN and CAN FD 82](#_Toc99383905)

[8.1.1 Addressing Format 82](#_Toc99383906)

[8.1.1.1 CAN 11-bit Identifiers 82](#_Toc99383907)

[8.1.2 Flow Control 83](#_Toc99383908)

[8.1.2.1 FlowStatus (FS) parameter 83](#_Toc99383909)

[8.1.3 Maximum Number of FC.Wait Frame Transmission (N\_WFTmax) 83](#_Toc99383910)

[8.1.3.1 BlockSize (BS) Parameter 83](#_Toc99383911)

[8.1.3.2 SeparationTime (STmin) parameter 84](#_Toc99383912)

[8.1.4 Timing Parameters 84](#_Toc99383913)

[8.1.5 Buffer Size 84](#_Toc99383914)

[8.1.6 TX\_DL 84](#_Toc99383915)

[8.2 Ethernet 85](#_Toc99383916)

[9 Data Link Layer 86](#_Toc99383917)

[9.1 CLASSICAL CAN Data Length Code (DLC) 86](#_Toc99383918)

[9.2 CAN FD Data Length Code (DLC) 86](#_Toc99383919)

[9.3 CLASSICAL CAN and CAN FD 86](#_Toc99383920)

[10 Explicit Diagnostic Gateways 87](#_Toc99383921)

[10.1 Explicit Gateway State Access 87](#_Toc99383922)

[10.2 Explicit Gateway State Access Not Active 87](#_Toc99383923)

[10.3 Entering Explicit Gateway State 88](#_Toc99383924)

[10.4 File Download Sequence to Sub-node Using Explicit Gateway 89](#_Toc99383925)

[11 Non-Bootloader ECUs 90](#_Toc99383926)

[11.1 Service 34H Programmable ECU via Application 90](#_Toc99383927)

[11.2 Non-Programmable ECU or Non Service 34H Programmable ECU 91](#_Toc99383928)

[Annex A : Example Link Traffic 93](#_Toc99383929)

[A.1 File Download 93](#_Toc99383930)

[A.2 File Upload 98](#_Toc99383931)

[Annex B : Code Examples 103](#_Toc99383932)

[B.1 CRC16-CITT C-code Example 1 (Fast) 103](#_Toc99383933)

[B.2 CRC16-CITT C-code Example 2 (Slow) 105](#_Toc99383934)

[Annex C : Example of How to Enter the Bootloader (PBL) From the Application 107](#_Toc99383935)

[Annex D : Conversion of All or Part of an ECU From Programmable to Masked ROM 110](#_Toc99383936)

[Annex E : ECU Programming Possibilities and Identification Overview 111](#_Toc99383937)

[E.1 ECUs on Main Public Network 111](#_Toc99383938)

[E.1.1 Implementation Details 111](#_Toc99383939)

[E.1.2 Identification Details 111](#_Toc99383940)

[E.2 ECUs on Transparent CAN Sub-Network 113](#_Toc99383941)

[E.2.1 Implementation Details 113](#_Toc99383942)

[E.2.2 Identification Details 113](#_Toc99383943)

[E.2.3 Sub Node Supports Diagnostics for Programming Only 113](#_Toc99383944)

[E.3 Sub Micros within ECU on a Public Main Network 115](#_Toc99383945)

[E.3.1 Implementation Details 115](#_Toc99383946)

[E.3.2 Identification Details 115](#_Toc99383947)

[E.4 ECUs on Public Sub-Network Connected Via Explicit Gateway 116](#_Toc99383948)

[E.4.1 Implementation Details 116](#_Toc99383949)

[E.4.2 Identification Details 116](#_Toc99383950)

[E.4.3 Sub Node Supports Diagnostics for Programming Only 116](#_Toc99383951)

[E.5 ECUs on Private Sub-Network (Not a Valid Implementation) 118](#_Toc99383952)

[E.5.1 Implementation Details 118](#_Toc99383953)

[E.5.2 Identification Details 118](#_Toc99383954)

[E.5.3 Issues with Approach 119](#_Toc99383955)

[Annex F : Inactive Memory Architecture Types 120](#_Toc99383956)

[F.1 Inactive Memory Architecture Type 1 – Hardware Facilitated Address Remapping 120](#_Toc99383957)

[F.2 Inactive Memory Architecture Type 2 –Memory Caching Option 1 120](#_Toc99383958)

[F.3 Inactive Memory Architecture Type 3 – Memory Caching Option 2 121](#_Toc99383959)

[F.4 Inactive Memory Architecture Type 4 – Execute from RAM 121](#_Toc99383960)

# List of figures

[Figure 2.1 Typical ECU memory map 21](#_Toc99383961)

[Figure 3.1 SWash Calculation 49](#_Toc99383962)

[Figure 4.1 Recommended service/end-of-line programming sequence (1 of 3) 61](#_Toc99383963)

[Figure 4.2 Recommended service/end-of-line programming sequence (2 of 3) 62](#_Toc99383964)

[Figure 4.3 Recommended service/end-of-line programming sequence (3 of 3) 63](#_Toc99383965)

[Figure 4.4 Minimum programming sequence (1 of 2) 64](#_Toc99383966)

[Figure 4.5 Minimum programming sequence (2 of 2) 65](#_Toc99383967)

[Figure 4.6 Download Data Block (1 of 2) 66](#_Toc99383968)

[Figure 4.7 Download Data Block (2 of 2) 67](#_Toc99383969)

[Figure 5.1 File upload procedure (1 of 5) 70](#_Toc99383970)

[Figure 5.2 File upload procedure (2 of 5) 71](#_Toc99383971)

[Figure 5.3 File upload procedure (3 of 5) 71](#_Toc99383972)

[Figure 5.4 File upload procedure (4 of 5) 72](#_Toc99383973)

[Figure 5.5 File upload procedure (5 of 5) 73](#_Toc99383974)

[Figure 6.1 Program mode 75](#_Toc99383975)

[Figure 6.2 Boot sequence 76](#_Toc99383976)

[Figure 11.1 Preparation by the application before entering the bootloader 107](#_Toc99383977)

[Figure 11.2 Bootloader program flow 108](#_Toc99383978)

# List of tables

[Table 1.1 References 15](#_Toc99383979)

[Table 1.2 – Software download services 18](#_Toc99383980)

[Table 2.1 – Primary bootloader services 22](#_Toc99383981)

[Table 2.2 – Secondary bootloader services 23](#_Toc99383982)

[Table 3.1 - DiagnosticSessionControl request message flow 25](#_Toc99383983)

[Table 3.2 - DiagnosticSessionControl positive response message flow 25](#_Toc99383984)

[Table 3.3 - ECUReset request message flow 26](#_Toc99383985)

[Table 3.4 - ECUReset positive response message flow 26](#_Toc99383986)

[Table 3.5 – SecurityAccess request message flow 27](#_Toc99383987)

[Table 3.6 – SecurityAccess positive response message flow 27](#_Toc99383988)

[Table 3.7 – SecurityAccess request message flow 28](#_Toc99383989)

[Table 3.8 – SecurityAccess positive response message flow 28](#_Toc99383990)

[Table 3.9 - ReadDataByIdentifier F111H request message flow 29](#_Toc99383991)

[Table 3.10 - ReadDataByIdentifier F111H positive response message flow 29](#_Toc99383992)

[Table 3.11 - ReadDataByIdentifier F113H request message flow 31](#_Toc99383993)

[Table 3.12 - ReadDataByIdentifier F113H positive response message flow 31](#_Toc99383994)

[Table 3.13 – ReadDataByIdentifier F18CH request message flow 31](#_Toc99383995)

[Table 3.14 - ReadDataByIdentifier F18CH positive response message flow 32](#_Toc99383996)

[Table 3.15 - ReadDataByIdentifier F180H request message flow 32](#_Toc99383997)

[Table 3.16 - ReadDataByIdentifier F180H positive response message flow 32](#_Toc99383998)

[Table 3.17 - ReadDataByIdentifier F109H request message flow 33](#_Toc99383999)

[Table 3.18 - ReadDataByIdentifier F109H positive response message flow 33](#_Toc99384000)

[Table 3.19 - ReadDataByIdentifier D100H request message flow 33](#_Toc99384001)

[Table 3.20 - ReadDataByIdentifier D100H positive response message flow 34](#_Toc99384002)

[Table 3.21 - ReadDataByIdentifier F162H request message flow 34](#_Toc99384003)

[Table 3.22 - ReadDataByIdentifier F162H positive response message flow 34](#_Toc99384004)

[Table 3.23 - ReadDataByIdentifier D028H request message flow 34](#_Toc99384005)

[Table 3.24 - ReadDataByIdentifier D028H positive response message flow 35](#_Toc99384006)

[Table 3.25 - ReadDataByIdentifier D03FH request message flow 35](#_Toc99384007)

[Table 3.26 - ReadDataByIdentifier D03FH positive response message flow 35](#_Toc99384008)

[Table 3.27 - ReadDataByIdentifier F17FH request message flow 35](#_Toc99384009)

[Table 3.28 - ReadDataByIdentifier F17FH positive response message flow 36](#_Toc99384010)

[Table 3.29 - WriteDataByIdentifier F111H request message flow 37](#_Toc99384011)

[Table 3.30 - WriteDataByIdentifier F111H positive response message flow 37](#_Toc99384012)

[Table 3.31 - WriteDataByIdentifier F113H request message flow 38](#_Toc99384013)

[Table 3.32 - WriteDataByIdentifier F113H positive response message flow 38](#_Toc99384014)

[Table 3.33 - WriteDataByIdentifier F18CH request message flow 39](#_Toc99384015)

[Table 3.34 - WriteDataByIdentifier F18CH positive response message flow 39](#_Toc99384016)

[Table 3.35 - WriteDataByIdentifier F17FH request message flow 40](#_Toc99384017)

[Table 3.36 - WriteDataByIdentifier F17FH positive response message flow 40](#_Toc99384018)

[Table 3.37 - RoutineControl 0300H request message flow, startRoutine 41](#_Toc99384019)

[Table 3.38 - RoutineControl 0300H request message flow, stopRoutine 41](#_Toc99384020)

[Table 3.39 - RoutineControl 0300H request message flow, requestRoutineResults 41](#_Toc99384021)

[Table 3.40 - RoutineControl 0300H positive response message flow, startRoutine 42](#_Toc99384022)

[Table 3.41 - RoutineControl 0300H positive response message flow, stopRoutine 42](#_Toc99384023)

[Table 3.42 - RoutineControl 0300H positive response message flow, requestRoutineResults 42](#_Toc99384024)

[Table 3.43 - RoutineControl 0301H request message flow 43](#_Toc99384025)

[Table 3.44 - RoutineControl 0301H positive response message flow 43](#_Toc99384026)

[Table 3.45 - RoutineControl FF00H request message flow 44](#_Toc99384027)

[Table 3.46 - RoutineControl FF00H positive response message flow 44](#_Toc99384028)

[Table 3.47 - RoutineControl FF01H request message flow 45](#_Toc99384029)

[Table 3.48 - RoutineControl FF01H positive response message flow 45](#_Toc99384030)

[Table 3.49 - RoutineControl 0304H request message flow 46](#_Toc99384031)

[Table 3.50 - RoutineControl 0304H positive response message flow 46](#_Toc99384032)

[Table 3.51 - RoutineControl 0305H request message flow 46](#_Toc99384033)

[Table 3.52 - RoutineControl 0305H positive response message flow 47](#_Toc99384034)

[Table 3.53 - RoutineControl 021BH request message flow 47](#_Toc99384035)

[Table 3.54 - RoutineControl 021BH positive response message flow 48](#_Toc99384036)

[Table 3.55 - RoutineControl 021AH request message flow 48](#_Toc99384037)

[Table 3.56 - RoutineControl 021AH positive response message flow 49](#_Toc99384038)

[Table 3.57 - RoutineControl 0219H request message flow 50](#_Toc99384039)

[Table 3.58 - RoutineControl 0219H positive response message flow 50](#_Toc99384040)

[Table 3.59 - RoutineControl 021CH request message flow 51](#_Toc99384041)

[Table 3.60 - RoutineControl 021CH positive response message flow 52](#_Toc99384042)

[Table 3.61 – RoutineType 52](#_Toc99384043)

[Table 3.62 – RoutineStatus 52](#_Toc99384044)

[Table 3.63 - RequestDownload request message flow 53](#_Toc99384045)

[Table 3.64 - RequestDownload positive response message flow 54](#_Toc99384046)

[Table 3.65 - RequestUpload request message flow 54](#_Toc99384047)

[Table 3.66 - RequestUpload positive response message flow 55](#_Toc99384048)

[Table 3.67 - TransferData request message flow (download) 55](#_Toc99384049)

[Table 3.68 - TransferData request message flow (upload) 56](#_Toc99384050)

[Table 3.69 - TransferData positive response message flow (download) 56](#_Toc99384051)

[Table 3.70 - TransferData positive response message flow (upload) 56](#_Toc99384052)

[Table 3.71 - RequestTransferExit request message flow 58](#_Toc99384053)

[Table 3.72 - RequestTransferExit positive response message flow 58](#_Toc99384054)

[Table 3.73 - TesterPresent request message flow 59](#_Toc99384055)

[Table 3.74 - TesterPresent positive response message flow 59](#_Toc99384056)

[Table 3.75 - Service not supported negative response message flow 59](#_Toc99384057)

[Table 7.1 - Diagnostic service timing parameters 79](#_Toc99384058)

[Table 7.2 - Requirements on maximum value of P4. 79](#_Toc99384059)

[Table 8.1 - 11-bit CAN identifiers 83](#_Toc99384060)

[Table 8.2 - Network layer timeout and performance requirement values 84](#_Toc99384061)

[Table 11.1 – Standard Software Identification DIDs 111](#_Toc99384062)

# Introduction

This document supplements the Ford Motor Company Production Purchasing Global Terms and Condition (“PPGTCs”) which are referenced in the Purchase Order. Any conflicts between this document and the PPGTCs shall be governed by the PPGTCs, unless specifically agreed in writing by an authorized Ford Motor Company representative (e.g., Purchasing buyer).

Electronic Control Units (ECUs or electronic modules) have demonstrated continued increase in feature, count, memory size, I/O circuit count, and part number complexity. All areas of the vehicle electrical/electronic system are affected, particularly powertrain and body computers. Electronic Control Units are experiencing late and frequent changes many of which affect software. These trends will undoubtedly continue into the foreseeable future.

There is a need to reduce vehicle build and service complexity, reduce ECU supplier complexity, and allow for maximum flexibility in software design by establishing designs and procedures providing the ability to configure and program ECUs in assembly and in service. Configuration and module programming will allow module design and release activities to minimize part complexity, economies of scale and allow bookshelf software to be utilized.

## Purpose/Scope

This document is intended to provide guidance and requirements to architects, system designers, application developers, and suppliers that are to deliver diagnostics compliant with Ford production vehicle programs. It shall serve as the cross-car line compliance reference for implementation of diagnostics in all Ford and Lincoln vehicles. Adherence to the specifications contained within this document is required to ensure compatibility within the corporate product line from vehicle to vehicle, model year to model year and maintain compatibility with the industry driven standards.

**NOTE:** Ford reserves the right to examine all related hardware and software designs related to this document for validation and verification of compliance.

## Target

Allow any microprocessor-based ECU to be fully programmed with strategy and calibrations at any assembly plant or dealership, over a Ford approved diagnostic communication link.

The following requirement principles shall be met for software download to an ECU:

* The ECU shall be accessible through a Ford approved diagnostic communication link.
* The ECU shall have non-volatile memory (e.g., flash memory or EEPROM) for software modifications.

The following guideline principles shall be met for software download to an ECU:

* When explicit agreements are made with the appropriate personnel at End of Line (EOL), ECUs are to be programmed fully or partly at EOL, with the goal of adding minimal cycle time to the assembly process.

It is recommended that all applicable hardware and software information stored in an ECU be recorded in a Vehicle Database before the completed vehicle leaves the factory. A Vehicle Database keeps a record of the vehicle configuration, which can be retrieved using the VIN, or other identification number/code. The Vehicle Database should always reflect the latest update of the vehicle.

## Use of This document

This document is to be used by ECU developers and any other entity utilizing the software download capabilities specified or referenced in this document. The requirements in this document apply primarily to when the ECU is executing the programmingSession. Requirements that apply when the ECU is executing the defaultSession or extendedDiagnosticSession within its normal application are described in [Ford GGDS] and may impose differences in terms of network layer timing, application layer timing, etc. Refer to section 1.9.1 for more details regarding requirements that are based upon level of ECU programmability.

Though an attempt was made to cover all aspects of implementing software there may still be design attributes that may lack detail or are overlooked. As these and other attributes are further studied requirements will be added or modified to reflect the latest industry and corporate directions and trends.

In the case of any requirement conflicting between either this document or any of the referenced documents, this document shall be the governing document. All requirements specified in this document shall supersede any requirement specified in any other document, except when conflicting with a legislated requirement (e.g., OBD-II).

## Traceability

These requirements shall include the requirements specified in [Ford GGDS], [Ford DCES], [Ford ASRS] and [Ford DPSA].

## Verification Method

These requirements shall be validated according to the test cases specified in [Ford DS Tests], [Ford DC Tests], [Ford MDX Tests] and [Ford DVP&R]. Instructions for downloading, installing and running validation tools along with how to analyze results are available in [Ford DVQStart].

## References

The following documents are either referenced by this specification or contain information that is relevant to this specification. Note that for all dated ISO or SAE specifications in Table 1.1. that are in a prepublication state (e.g., FDIS status), any available version between the dated version and the first official publication following this dated version are acceptable for usage.

Table 1.1 References

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Reference | Source | Title | Version or date | Document Number |
| [ISO 11898-1] | ISO | Road vehicles — Controller area network (CAN) — Part 1: Data link layer and physical signalling | 2015 | [ISO 11898-1](https://www.iso.org/standard/63648.html) |
| [ISO 13400-2] | ISO | Road Vehicles – Diagnostic communication Over Internet Protocol (DoIP) – Part 2: Transport Protocol and Network layer Services | 2019 | [ISO 13400-2](https://www.iso.org/obp/ui/#iso:std:iso:13400:-2:ed-2:v1:en) |
| [ISO 14229-1] | ISO | Road vehicles – Unified diagnostic services (UDS) – Part 1: Specification and requirements | 2020 | [ISO 14229-1](https://www.iso.org/standard/45763.html) |
| [ISO 14229-2] | ISO | Road vehicles – Unified diagnostics services (UDS) – Part 2: Session layer services | 2021 | [ISO 14229-2](https://www.iso.org/standard/45763.html) |
| [ISO 14229-3] | ISO | Road vehicles – Unified diagnostics services (UDS) – Part 3: Unified diagnostic services on CAN implementation (UDSonCAN) | 2012 | [ISO 14229-3](https://www.iso.org/standard/55284.html) |
| [ISO 14229-5] | ISO | Road Vehicles – Unified Diagnostic Services (UDS) – Part 5: Unified Diagnostic Services on Internet Protocol Implementations (UDSonIP) | 2013 | [ISO 14229-5](https://www.iso.org/standard/55287.html) |
| [ISO 15031-3] | ISO | Road vehicles – Communication between vehicle and external test equipment for emission-related diagnostics – Part 3: Diagnostic connector and related electrical circuits, specification and use | 2016 | [ISO 15031-3](https://www.iso.org/standard/64636.html) |
| [ISO 15765-2] | ISO | Road vehicles – Diagnostic communication over Controller Area Network (DoCAN) – Part 2: Transport protocol and network layer services | 2016 | [ISO 15765-2](https://www.iso.org/standard/66574.html) |
| [Ford GGDS] | Ford | Generic Global Diagnostic Specification, Part 1 Diagnostic Implementation Requirements | Latest | [VDOC140354](https://www.vsemawc.ford.com/awc/#/com.siemens.splm.clientfx.tcui.xrt.showObject?uid=SyqJCanmx3NrTD&spageId=Overview) |
| [Ford DoIP] | Ford | Ford Diagnostics over Internet Protocols (DoIP) Specification | Latest | [VDOC139849](https://www.vsemawc.ford.com/awc/#/com.siemens.splm.clientfx.tcui.xrt.showObject?uid=D8iJxNK5x3NrTD&spageId=Overview) |
| [Ford DCES] | Ford | Data Compression and Encryption Specification | Latest | [VDOC010450](https://www.vsemawc.ford.com/awc/#/com.siemens.splm.clientfx.tcui.xrt.showObject?uid=FxcNTTh9x3NrTD) |
| [Ford ASRS] | Ford | Application Signing Requirements Specification | Latest | [VDOC076733](https://www.vsemawc.ford.com/awc/#/com.siemens.splm.clientfx.tcui.xrt.showObject?uid=x%24Zx8BcYx3NrTD) |
| [Ford CAN DL&PHY] | Ford | CAN Data Link and Physical Layer | Latest | [VDOC006688](https://www.vsemawc.ford.com/awc/#/com.siemens.splm.clientfx.tcui.xrt.showObject?uid=8iWJpzWLx3NrTD) |
| [FORD\_VPM] | Ford | Variable Power Moding Functional Specification | Latest | [F003250](https://www.vsemawc.ford.com/awc/#/com.siemens.splm.clientfx.tcui.xrt.showObject?uid=CvX5H2Shx3NrTD) |
| [Ford DPSA] | Ford | EESE Diagnostic Programming Security Algorithm  Note: This document is Confidential and not shared with ECU and test tool suppliers. Instead, ECU and test tool suppliers are provided access to developed software components that implement these requirements. | 001 | N/A |
| [Ford GMRDB] | Ford | Global Main Reference Database (ISO 14229-1 based) | N/A | [VSEM GMRDB](https://www.vsemawc.ford.com/awc/#/com.siemens.splm.clientfx.tcui.xrt.showObject?uid=gaaVc7Gix3NrTD) |
| [Ford IVS] | Ford | In Vehicle Software | N/A |  |
| [Ford DVQStart] | Ford | Ford Diagnostic Validation – Overview and Quick Start Guide | Latest | [VDOC011034](https://www.vsemawc.ford.com/awc/#/com.siemens.splm.clientfx.tcui.xrt.showObject?uid=hCbNlSimx3NrTD) |
| [Ford GGDS Tools] | Ford | Diagnostic Application Note: Recommended Practice for GGDS Tools | Latest | [VDOC011033](https://www.vsemawc.ford.com/awc/#/com.siemens.splm.clientfx.tcui.xrt.showObject?uid=hhQNlSimx3NrTD) |
| [Ford DS Tests] | Ford | GGDS and SWDL Diagnostic Services Test Specification 006 | Latest | [VDOC139806](https://www.vsemawc.ford.com/awc/#/com.siemens.splm.clientfx.tcui.xrt.showObject?uid=eQkJRl28x3NrTD) |
| [Ford DC Tests] | Ford | GGDS and SWDL Diagnostic Communication Test Specification 006 | Latest | [VDOC139805](https://www.vsemawc.ford.com/awc/#/com.siemens.splm.clientfx.tcui.xrt.showObject?uid=OwmJRl28x3NrTD) |
| [Ford SWDL Tests] | Ford | Software Download Functional Test Specification 008 | Latest | [VDOC139808](https://www.vsemawc.ford.com/awc/#/com.siemens.splm.clientfx.tcui.xrt.showObject?uid=esgJRl28x3NrTD) |
| [Ford DVP&R] | Ford | Diagnostic DVP&R for GGDS-006/SWDL-008 | Latest | [VDOC139812](https://www.vsemawc.ford.com/awc/#/com.siemens.splm.clientfx.tcui.xrt.showObject?uid=uEpJRl28x3NrTD) |
| [Ford MDX Tests] | Ford | Multiplex Diagnostic Exchange Validation Specification | Latest | [VDOC011036](https://www.vsemawc.ford.com/awc/#/com.siemens.splm.clientfx.tcui.xrt.showObject?uid=RyRNle8wx3NrTD) |

## Abbreviations, Acronyms and Definitions

The following abbreviations are used throughout this specification:

|  |  |
| --- | --- |
| **Terms** | **Acronyms or Definitions** |
| A/B or A/B/A | A/B and A/B/A both refer to inactive memory architecture types where an ECU supports the ability to program a new strategy (e.g., B) without affecting the existing strategy (e.g., A). Annex F provides high level examples. A/B generally refers to the scenario where the memory is doubled what it would be without an inactive partition and A/B/A generally refers to the scenario where the memory is tripled and a separate active backup is required in case of a rollback. |
| CAN | Controller Area Network |
| CAN FD | **C**ontroller **A**rea **N**etwork **F**lexible **D**ata rate  Controller area network with flexible data rate and larger payload as defined in [ISO 11898-1]. |
| CLASSICAL CAN | Controller area network with static data rate and up to eight data bytes as defined in [ISO 11898-1]. |
| Diagnostic Feature Owner | Person responsible for the diagnostic requirements / coverage of the ECU. This person is ultimately responsible for ensuring that the diagnostics (e.g., the details of the fault detection logic for each DTC) are correct and adequate (e.g., DTCs get set only after confirming the presence of a fault and a customer noticeable problem). This is typically the Ford D&R and/or subject matter expert for the particular ECU, subsystem, and/or feature. |
| DLC | Data Link Connector or **D**ata **L**ength **C**ode (CAN frame) |
| DoIP | Diagnostics over Internet Protocols |
| ECU | Electronic Control Unit |
| EOL | End of Line |
| Explicit Diagnostic Gateway | An ECU which connects a public CAN network to a sub-network in order to allow a tester to perform diagnostics on the sub-network ECUs. This approach requires explicit tester knowledge of the sub-network in order to perform most diagnostic tasks. The sub-network protocol is not limited to CAN. See section 10. |
| FCSD | Ford Customer Service Division |
| FMC | Ford Motor Company |
| HW | Hardware |
| Implementer | Person responsible for the actual diagnostic implementation of the ECU. It may be the vehicle manufacturer or the ECU supplier diagnostic software designer. |
| IVS | In Vehicle Software |
| Logical Block | A software unit that consists of a predefined programmable contiguous memory areas as defined in [Ford ASRS] |
| M | Mandatory |
| Main network | A network connected to the vehicle diagnostic connector which is partly or entirely used for diagnostic communications |
| Main node | An ECU that is connected to a main network. The ECU may or may not include a diagnostic gateway. |
| N/A | Not Applicable |
| O | Optional |
| Off-Board Tester | Testers which are only attached temporarily to the vehicle at the DLC for a particular task, and then are disconnected after that task. |
| On-Board Tester | Testers which remain connected to one or more of the vehicle's communication networks when the vehicle is in the hands of the customer/driver that sends diagnostic requests and receives diagnostic responses. This tester may be permanently attached or may be attached through the DLC. |
| PBL | Primary bootloader |
| Private network | A communication network that is not connected to the vehicle diagnostic connector (either directly or through a diagnostic gateway). Note that this definition applies to this specification and is not necessarily applicable to any other specification. |
| Public network | Main or sub-network |
| Root Hash | Hash calculated over a single verification structure used to generate a digital signature as defined in [Ford ASRS] |
| SBL | Secondary bootloader |
| SID | Service identifier |
| SPRMIB | SuppressPositiveResponseMessageIndicationBit |
| SSDS | Subsystem Specific Diagnostic Specification (Part 2 Diagnostic Specification) |
| Sub-network | A network connected to the main network via a diagnostic gateway which is partly or entirely used for diagnostic communications. |
| Sub-node | An ECU that is connected to a sub-network. |
| SW | Software |
| SWash | Software Hash of specific logical blocks |
| System Supplier Specific | Ranges of parameters reserved within reference [ISO 14229-1] which may be defined by the ECU supplier. These parameter values are intended solely for supplier usage and must not be required for usage by official Ford manufacturing or service tools. |
| TBD | To be defined |
| Tester | A device used to connect to the vehicle networks that sends diagnostic requests and receives diagnostic responses (e.g. workshop electronic tool, manufacturing equipment). |
| Transparent CAN Diagnostic Gateway | An ECU which connects a public CAN network to a CAN sub-network in order to allow a tester to perform diagnostics on the sub-network ECUs. This is achieved by the gateway always relaying raw diagnostic request and response CAN frames in a manner that does not require any knowledge of the sub-network existence to the tester. The sub-network protocol is limited to CAN. See [Ford GGDS]. |
| TX\_DL | **T**ransmit **D**ata **L**ength (in bytes)  Configures the maximum usable payload length in bytes of the data link layer in the transmitter for the application that implements the network layer. |
| VIN | Vehicle identification number |

## File Formats

This specification is completely independent of which file formats are to be used in the Vehicle Database and in the tester for storage of the ECU software.

## Overview

Software Download involves the transfer of information from a tester to an ECU using data normally stored in an officially released file. The data transferred may contain calibration data for the ECU or completely new software for the ECU. It also includes uploading of data from the ECU to the tester for debugging.

The Software Download concept utilizes the following ISO specifications as a base:

* [ISO 14229-1] Describes the services used in the Software Download concept.
* [ISO 14229-2] Describes the session layer details relevant to the Software Download concept.

In [ISO 14229-1] the following services are relevant for software download (and upload):

Table 1.2 – Software download services

|  |  |
| --- | --- |
| Service | Use of service for software download |
| DiagnosticSessionControl | The tester requests the ECU to enter/exit the program mode |
| ECUReset | The tester requests the ECU to perform a reset (and therefore exit the program mode) |
| SecurityAccess | Unlock the ECU for download/upload |
| ReadDataByIdentifier | The tester requests the ECU to report data (e.g., identification information) |
| WriteDataByIdentifier | Write ECU data (e.g., core assembly part number, serial number) |
| RoutineControl | Activate the secondary bootloader, erase flash memory, calculate checksum, enter/exit explicit gateway state, ... |
| RequestDownload | Request download to ECU memory (e.g., RAM, flash, or EEPROM) |
| RequestUpload | Request upload from ECU memory (e.g., RAM, flash, or EEPROM) |
| TransferData | Transfer data to/from ECU memory (e.g., RAM, flash, or EEPROM) |
| RequestTransferExit | Exit transfer of data to/from ECU memory |
| TesterPresent | The tester indicates to the ECU(s) that it is still present |

### Levels of Programmability

There are basically five variants of ECU architecture when determining the level of programmability per this specification:

**ECU Variant #1:** **Service 34H Programmable ECU via a bootloader**

This is the typical and desired solution when an ECU supports the capability of performing software download. When an ECU is completely programmable (i.e., it is possible to program the entire ECU's application and calibrations) or when the ECU supports programming of flash memory, this is the required solution. When this solution is implemented, all requirements within this specification apply, except for those in section 0.

**ECU Variant #2: Service 34H Programmable ECU via a "permanent" (e.g., masked ROM) application**

This is a rare non-typical solution in which an ECU only supports the capability of performing a limited software download. This solution is only allowed when an ECU is partially programmable and therefore always has a fixed application which implements the programming capability. An example of this use case is a radio which has a fixed application but needs to download audio equalizer data. When this solution is implemented on an ECU, the programming requirements are contained within section 11.1.

**ECU Variant #3: Non-programmable ECU**

This is the case where an ECU does not support software download at all (i.e., service 34H requestDownload is never supported) and is not programmable via Ford by other means. When this solution is implemented on an ECU, the programming requirements are contained within section 11.2.

**ECU Variant #4: Service 34H Programmable via a “replaceable” application**

This solution has been implemented by some Ford infotainment ECUs where the ECU supports A/B type memory to allow full programming of the entire application without transitioning to a primary bootloader. In this case, the new software is programmed while the ECU is still executing its’ application. When this solution is implemented on an ECU, the programming requirements are contained within section 11.1.

**ECU Variant #5: Autosar Adaptive ECU Programmable not using Service 34H**

This solution applies to Autosar Adaptive ECUs which are programmable via the Autosar Update and Configuration Manager mechanism not using service 34H. When this solution is implemented on an ECU, the programming requirements are contained within section 11.2.

## Deviations

A Subsystem Specific Diagnostic Specification (SSDS) shall be generated to capture the diagnostic implementation details for each ECU. Any non-compliance to this specification, for any ECU, is a deviation and shall follow the appropriate process for resolution. Any known deviations shall be fully documented and adequately explained in the SSDS for the ECU.

**NOTE**: This specification addresses only the use of diagnostic strategies, communications and other relevant specifications related to the diagnostic protocol. It does not address normal mode multiplex ECU to ECU communications used for normal operation messaging.

## Tester Support

A tester shall be capable of performing software download on an ECU designed to this specification using only the specific diagnostic services, data parameters, sub-functions, etc. specified within this specification. Any special ECU requirements which necessitate specific tester support over and beyond what is documented in this specification (e.g., execution of a special routine to complete the download to a sub-processor, specific wait times between requests, etc.) shall not be implemented unless explicit approval is granted by Ford Core Network Communications.

## Programming Voltages

An ECU shall not require that the tester supply an external voltage to perform any software download procedure.

## Diagnostic Communication Requirements

The diagnostic services required by this document shall conform to standards in accordance with [ISO 14229-1] and [ISO 14229-2] with Ford specific requirements defined in [Ford GGDS].

Where the ECU supports diagnostic services over CAN or CAN FD, the ECU shall conform to diagnostics over CAN standards in accordance with [ISO 14229-3] and [ISO 15765-2].

Where the ECU supports diagnostic services over internet protocols (DoIP), the ECU shall conform to diagnostic over IP standards in accordance with Ford specific requirements defined in [Ford DoIP].

No other diagnostic communication is allowed unless it is approved by Ford Core Network Communications.

# Two Level Bootloader

This section further defines the details for programmable ECUs via the bootloader (see Variant #1 in section 1.9.1). The bootloader is divided into two separate and mandatory parts: a primary bootloader (PBL) and a secondary bootloader (SBL). The PBL is designed to be permanently placed in a protected flash boot sector, which is activated from reset. The SBL is intended to be downloaded by the PBL into the CPU internal RAM, from where it is executed and thereafter deleted after each use. The SBL can be described as a superset of the PBL, adding functions for erase and program of flash memory and EEPROM.



Figure 2.1 Typical ECU memory map

## Primary Bootloader (PBL)

The primary bootloader (PBL) code size shall be minimized such that it can easily fit into a boot flash sector.

The PBL shall be error free. It is programmed into a protected flash boot sector that cannot be updated once an ECU is released in production.

The PBL shall only be capable of writing data to RAM due to security and data integrity. An erase of the flash memory can only occur if the secondary bootloader (SBL) is downloaded into RAM.

The memory area containing the PBL shall be protected from erasure to eliminate the possibility of accidentally erasing it, but if the processor architecture permits this protection to be configured such that it is removable, then the protection shall be set up such that it could be removed if a special SBL were to be written for this purpose.

The PBL shall not be able to overwrite or modify any of its own memory.

The standard SBL shall not be able to overwrite or erase the PBL. Please note that this specification does not require any such special SBL to be implemented or provided.

The PBL shall provide functions to identify the ECU hardware (e.g., core assembly part number, etc.).

When an ECU transitions to the programmingSession, the primary bootloader shall pre-condition all of the ECU's I/O into a state where components cannot be damaged or be in a state that is not safe for people working on the vehicle.

### Primary Bootloader Services

The following table determines what services are mandatory or optional.

Table 2.1 – Primary bootloader services

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Service | | | Man/Opt (See Note 1) | | Description |
| default-Session | programming-Session |
| 10H | DiagnosticSessionControl | | M | M | Enter program mode |
| 11H | ECUReset | | M | M | Exit program mode |
| 27H | SecurityAccess | | N/A | M | Unlock the ECU |
| 22H | ReadDataByIdentifier | | | | Identify the ECU |
|  | F111H | ECU Core Assembly Number | M | M |
|  | F113H | ECU Delivery Assembly Number | M | M |
|  | F18CH | ECU Serial Number | M | M |
|  | F17FH | Ford Electronic Serial Number | MC5 | MC5 |
|  | F180H | Boot Software Identification | MC1 | MC1 |
|  | F109H | Boot Software Version Number | MC1 | MC1 |
|  | D100H | Active Diagnostic Session | M | M |
|  | F162H | Software Download Specification Version | M | M |
|  | D028H | Application Signature Validation Status | MC3 | MC3 | Signature feedback |
|  | D03FH | In-Use Application Signing Public Key Hash | MC3 | MC3 | Public key hash for signing |
| 31H | RoutineControl | | | | Activate SBL, etc. |
|  | 0300H | Gateway State Access | N/A | MC2 |
|  | 0301H | Activate Secondary Bootloader | N/A | M |
|  | FF01H | CheckProgrammingDependencies  (See Note 2) | N/A | M |
|  | 0304H | Check Valid Application  (See Note 2) | N/A | M |
|  | 021BH | Set OTA Update State | N/A | MC4 | Set which partition an update will utilize |
|  | 0219H | Calculate SWash | N/A | MC4 | Calculate software hash |
| 34H | RequestDownload | | N/A | M | Request download to RAM |
| 36H | TransferData | | N/A | M | Transfer data to RAM |
| 37H | RequestTransferExit | | N/A | M | Exit transfer to RAM |
| 3EH | TesterPresent | | M | M | The tester indicates to the ECU(s) that it is still present |

Abbreviations in table:

|  |  |  |
| --- | --- | --- |
| MC1 | = | At least one of these dataIdentifiers shall be supported by the ECU |
| MC2 | = | Mandatory if the ECU supports the full software download explicit diagnostic gateway (see section 10) |
| MC3 | = | Mandatory if the ECU supports code signing |
| MC4 | = | Mandatory if the ECU supports an inactive memory partition (e.g., A/B or A/B/A type approach) |
| MC5 | = | Mandatory if required by the ECU to support signed service requests |

Notes in table:

1. Note that the defaultSession within the primary bootloader is **not** the same as the defaultSession within the application. The primary bootloader defaultSession is only accessible upon boot up when the ECU has determined that no valid application is present (see section 6.1 for more details).
2. Under special circumstances, Ford Core Network Communications may authorize the non-support of this routine in the PBL. The routine shall still always be supported in the SBL.

## Secondary Bootloader (SBL)

It should be clear that there is a need for special routines for flash EEPROM erasure and programming. These routines will be different for every ECU family depending on the hardware components and design. The SBL includes all routines for flash erase and flash program of data to an ECU.

The SBL is downloaded with the help of the PBL into RAM. After download the SBL is activated from the PBL. The SBL services are now added (super set to the PBL) and can be used together with the PBL services. This means that all PBL services shall be capable of being executed after the SBL has been downloaded and activated. Refer to section 4 and/or section 5 for details regarding how and when the SBL is downloaded and activated.

The preferred implementation is that the SBL also includes all PBL services. The advantage is that if an error is found in the PBL, it is possible to make a workaround solution in the SBL. Another advantage is that no calls are necessary from the SBL to the PBL, which simplify the bootloader implementation. The disadvantage is that the SBL will use more RAM space and it will take slightly more time to download the SBL.

When the bootloader concept is implemented, the support of a secondary bootloader is mandatory. Under special circumstances, Ford Core Network Communications may authorize the non-support of a secondary bootloader so long as this method can be proven to provide equivalent or near equivalent functionality and has no effect on the integrity of the ECU or the file download sequence described in section 4 (i.e., the implementation shall be transparent to the tester other than no longer requiring the need for a download and activation of the secondary bootloader).

### Secondary Bootloader Services

The following table determines what services are mandatory or optional.

Table 2.2 – Secondary bootloader services

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Service** | | | **Man/Opt** | **Description** |
| **Programming-Session** |
| 2EH | WriteDataByIdentifier | | | Write ECU ­data (e.g., core assembly part number, serial number) |
|  | F111H | ECU Core Assembly Number | O |
|  | F113H | ECU Delivery Assembly Number | O |
|  | F18CH | ECU Serial Number | O |
|  | F17FH | Ford Electronic Serial Number | O |
| 31H | RoutineControl | | |  |
|  | 021AH | Prepare Inactive Memory | MC1 |  |
| 021CH | Replace Active Memory Partition | MC1 | Replace the active memory partition with either the inactive partition or the active backup. |
|  | FF00H | Erase Memory | M | Erase flash memory |
| 0305H | Update Security Bytes | O |  |
| 34H | RequestDownload | | M | Download to flash or EEPROM |
| 35H | RequestUpload | | O | Upload to RAM, flash, or EEPROM |
| 36H | TransferData | | M | Transfer data to flash or EEPROM |
| 37H | RequestTransferExit | | M | Exit transfer to flash or EEPROM |

Abbreviations in table:

|  |  |  |
| --- | --- | --- |
| MC1 | = | Mandatory if the ECU supports an inactive memory partition (e.g., A/B or A/B/A type approach) |

After download and start of execution (i.e., activation) of the SBL in RAM, it shall not be possible to make a new download to RAM (i.e., ECU shall reject request with NRC 31H). Before a new download to RAM can be performed, an exit from the programmingSession shall be required (see Figure 6.1) to avoid overwriting the SBL. A transition from programmingSession to programmingSession using service 10H (DiagnosticSessionControl) shall be considered an exit from programmingSession and allow for a new download into RAM as it must reset security access and reset any functionality dependent upon security access (see [ISO 14229-1]), including an activated SBL.

## Additional Bootloader Resources for Secondary Processor Elements

Some ECU hardware architectures may include additional microcontrollers (secondary processors). If these microcontrollers have programmable memory areas additional bootloader resource executing on the secondary processor is necessary. In these cases, the SBL will need to establish communication with the secondary processor.

This specification does not include any detail for this communication method and any such additional bootloader resource shall be defined outside of this specification. However, the following requirements shall be complied with:

* The download communication method implemented on the secondary processor shall be protected such that erasing any or all of the programmable memory areas of the secondary processor and removing the ECU power at any time shall not prevent the subsequent reprogramming of any normally programmable memory area. In addition, the complete or partial downloading of any non-operational or partly operational software into the programmable memory areas of the secondary processor shall not prevent the subsequent reprogramming of any normally programmable memory area

Refer to section E.3 for more details.

# Software Download Services

The ECU shall support diagnostic service requests in the programmingSession as detailed in this specification. All diagnostic service parameters not specifically called out in this specification shall be considered as never supported in the programmingSession, except for parameters labelled as SystemSupplierSpecific in [ISO 14229-1]. Note that the vehicle manufacturer’s tools will not support service parameters within the SystemSupplierSpecific ranges. The ECU is not allowed to support other diagnostic service requests unless it is approved by Ford Core Network Communications. The diagnostic services are defined in [ISO 14229-1]. Specific requirements for implementing the services on CAN are specified in [ISO 14229-3]. Specific requirements for implementing the services on Ethernet are specified in [Ford DoIP].

## DiagnosticSessionControl (10H) service

The DiagnosticSessionControl service and sub-function programmingSession shall be used for entering program mode.

Table 3.1 - DiagnosticSessionControl request message flow

|  |  |  |  |
| --- | --- | --- | --- |
| Message direction: | | Tester → ECU | |
| Message Type: | | Request | |
| Data byte | Description (all values are in hexadecimal) | | Byte Value (Hex) |
| #1 | DiagnosticSessionControl request SID | | 10 |
| #2 | diagnosticSessionType = programmingSession or defaultSession | | 02 or 82 or 01 or 81 |

The ECU shall enter program mode if diagnosticSessionType = programmingSession and suppressPosRspMsgIndicationBit = TRUE or FALSE.

The ECU shall perform a hard reset, equal to ECUReset(hardReset), if diagnosticSessionType = defaultSession and suppressPosRspMsgIndicationBit = TRUE or FALSE.

If diagnosticSessionType = defaultSession and suppressPosRspMsgIndicationBit = FALSE the sessionParameterRecord shall include the values for P2Server\_max = 50 ms and P2\*Server\_max = 5000 ms (i.e., the default values for the defaultSession per ref. [Ford GGDS]).

If diagnosticSessionType = programmingSession and suppressPosRspMsgIndicationBit = FALSE the sessionParameterRercord shall include the values P2Server\_max and P2\*Server\_max for the programmingSession.

Table 3.2 - DiagnosticSessionControl positive response message flow

| Message direction: | | ECU → Tester | |
| --- | --- | --- | --- |
| Message Type: | | Response | |
| Data byte | Description (all values are in hexadecimal) | | Byte Value (Hex) |
|  |  | |  |
| #1 | DiagnosticSessionControl response SID | | 50 |
| #2 | diagnosticSessionType = defaultSession or programmingSession | | 01 or 02 |
| #3 | sessionParameterRecord [byte 1] = P2Server\_max (high byte) | | 00-FF |
| #4 | sessionParameterRecord [byte 2] = P2Server\_max (low byte) | | 00-FF |
| #5 | sessionParameterRecord [byte 3] = P2\*Server\_max (high byte) | | 00-FF |
| #6 | sessionParameterRecord [byte 4] = P2\*Server\_max (low byte) | | 00-FF |

For more information regarding sessionParameterRecord, P2Server and P2\*Server see [ISO 14229-1] and [ISO 14229-2].

### Supported Negative Response Codes

See [ISO 14229-1] for DiagnosticSessionControl supported negative response codes.

## ECUReset (11H) service

The ECUReset service shall be used to exit the ECU programmingSession by performing a reset. The request message data parameter resetType shall be set to hardReset. The tester may request to suppress the positive response message by setting the suppressPosRspMsgIndicationBit (bit 7 of the sub-function parameter) to TRUE.

Table 3.3 - ECUReset request message flow

|  |  |  |  |
| --- | --- | --- | --- |
| Message direction: | | Tester → ECU | |
| Message Type: | | Request | |
| Data byte | Description (all values are in hexadecimal) | | Byte Value (Hex) |
| #1 | ECUReset request SID | | 11 |
| #2 | resetType = hardReset | | 01 or 81 |

The ECUReset service shall be performed if resetType = hardReset and suppressPosRspMsgIndicationBit = TRUE or FALSE.

Table 3.4 - ECUReset positive response message flow

| Message direction: | | ECU → Tester | |
| --- | --- | --- | --- |
| Message Type: | | Response | |
| Data byte | Description (all values are in hexadecimal) | | Byte Value (Hex) |
| #1 | ECUReset response SID | | 51 |
| #2 | resetType = hardReset | | 01 |

### Supported Negative Response Codes

See [ISO 14229-1] for ECUReset supported negative response codes.

## SecurityAccess (27H) service

The SecurityAccess service shall be implemented in all programmable ECUs to restrict access from unapproved tools and to unlock the ECU for download and upload of data. The security access service handling and procedure shall be fully compliant with Annex I from [ISO 14229-1]. The security algorithm associated with the requestSeed value of 01H shall be the Ford EESE Diagnostic Programming Security Algorithm [Ford DS Tests]. When using this algorithm, the Delay\_Timer and Att\_Cnt parameter values (see [ISO 14229-1], Annex I) shall not be supported and the Static\_Seed parameter shall have a value of false.

The security seed shall be randomly generated.

The security level secret key (fixed bytes) SHALL:

* Contain the number of bytes required by the implemented security algorithm.
* Be unique per security level.
* Be randomly generated independently per supported security level (e.g., do not randomly generate for one level in an ECU and use that same value in another level by changing a couple bytes).
* Be classified as FORD SECRET and managed accordingly.

The security level secret key SHALL NOT:

* Be shared across products or different types of ECUs. *(example: PCM and ABS module shall not share the same key* *but all PCM ECUs may share the same key and all ABS ECUs may share the same key)*
* Be stored unencrypted in the ECU.
* Contain 3 or more consecutive repeating bytes.   
  Example: 03H, 00H, 00H, 00H, 1AH, F8H, 09H, AAH, 0BH, C1H, 05H, 67H
* Consist of 12 sequential bytes.   
  Example: 03H, 04H, 05H, 06H, 07H, 08H, 09H, 0AH, 0BH, 0CH, 0DH, 0EH
* Spell words with hexadecimal values   
  Example: FAH, CAH, DEH, or DEH, ADH, or BEH, EFH
* Spell words with ASCII or Unicode characters   
  Example: 46H, 6FH, 72H, 64H, 53H, 65H, 63H, 72H, 65H, 74H, 4BH, 65H = “FordSecretKe” in ASCII
* Be stored as a text string.

Example: “1A2B457F8390CBA1706A113B”

* Use numerical data such as serial numbers, part numbers or phone numbers

The following diagnostic functionality shall never be capable of being executed in the programmingSession unless the security level corresponding to a requestSeed value of 01H is unlocked. All other diagnostic functionality contained within this specification shall not be protected by SecurityAccess.

* Service 2EH writeDataByIdentifier, Service 34H requestDownload, service 35H requestUpload, service 36H transferData, service 37H requestTransferExit
* Service 31H routineControl for the following routines
  + Routine 021CH Replace Active Memory Partition
  + Routine 021AH Prepare Inactive Memory
  + Routine 0301H Activate Secondary Bootloader
  + Routine 0305H Update Security Bytes
  + Routine FF00H Erase Memory

**NOTE**: Some diagnostic functionality may be implicitly blocked when the ECU is locked and therefore may not have a use case to respond with negative response code 33H (securityAccessRequired). For example, the ECU is required to respond with requestSequenceError if a transferData or requestTransferExit is received without an active requestDownload or requestUpload service.

The securityAccess service is primarily intended to prevent unauthorized writes to RAM (e.g., downloading of the SBL). ECUs shall still implement any necessary checks to ensure code that is downloaded to RAM is intended or authorized for the ECU. Refer to section 6.2 for more details on protection against unauthorized execution of strategy code.

### Step 1: Request the Seed

Table 3.5 – SecurityAccess request message flow

|  |  |  |  |
| --- | --- | --- | --- |
| Message direction: | | Tester → ECU | |
| Message Type: | | Request | |
| Data byte | Description (all values are in hexadecimal) | | Byte Value (Hex) |
| #1 | SecurityAccess request SID | | 27 |
| #2 | securityAccessType = requestSeed | | 01 or 81 |

Table 3.6 – SecurityAccess positive response message flow

| Message direction: | | ECU → Tester | |
| --- | --- | --- | --- |
| Message Type: | | Response | |
| Data byte | Description (all values are in hexadecimal) | | Byte Value (Hex) |
| #1 | securityAccess response SID | | 67 |
| #2 | securityAccessType = requestSeed | | 01 |
| #3 | securitySeed [byte 1] = seed #1 (high byte) | | 00-FF |
| : | : | | : |
| #n | securitySeed [byte m] = seed #m (low byte) | | 00-FF |

If the ECU is in an unlocked state for the requested security level the securitySeed bytes shall have the value of 00H.

Security level 01H (i.e., requestSeed 01H) shall only be used to protect access to software download related diagnostic functionality and shall only be supported in the programmingSession (i.e., different security levels shall be used if security access is required by the diagnostic application).

The security seed length (m) in Table 3.6 shall be fixed to the size defined in [Ford DPSA].

### Step 2: Send the Key

Table 3.7 – SecurityAccess request message flow

|  |  |  |  |
| --- | --- | --- | --- |
| Message direction: | | Tester → ECU | |
| Message Type: | | Request | |
| Data byte | Description (all values are in hexadecimal) | | Byte Value (Hex) |
| #1 | SecurityAccess request SID | | 27 |
| #2 | accessType = sendKey | | 02 or 82 |
| #3 | securityKey [byte 1] = key #1 (high byte) | | 00-FF |
| : | : | | : |
| #q | securityKey [byte p] = key #p (low byte) | | 00-FF |

Table 3.8 – SecurityAccess positive response message flow

| Message direction: | | ECU → Tester | |
| --- | --- | --- | --- |
| Message Type: | | Response | |
| Data byte | Description (all values are in hexadecimal) | | Byte Value (Hex) |
| #1 | SecurityAccess response SID | | 67 |
| #2 | accessType = sendKey | | 02 |

The key length (p) in Table 3.7 shall be fixed to the size defined in [Ford DPSA].

### Supported Negative Response Codes

See [ISO 14229-1] for SecurityAccess supported negative response codes.

## ReadDataByIdentifier (22H) service

The ReadDataByIdentifier service shall be used to identify the ECU hardware and software using specific dataIdentifiers. The ReadDataByIdentifier service shall support a minimum of one dataIdentifier in a single request. The format of all dataIdentifiers in this specification are detailed in [Ford GMRDB]. The data content of all dataIdentifiers shall conform to all rules as described in Annex E of [Ford GGDS]. The supplier shall ensure they have mechanisms in place to always report accurate values for all dataIdentifiers upon initial delivery to Ford.

Certain dataIdentifiers may also be read by the application (e.g., defaultSession and extendedDiagnosticSession) as specified in [Ford GGDS] as well as during the programmingSession using the identifiers required in this section. In these cases, reading the dataIdentifier in the defaultSession or extendedDiagnosticSession shall read data from the same storage location or by using the same common function or calculation routine as is used for reading the dataIdentifier in the programmingSession.  All dataIdentifiers implemented in the programmingSession shall be capable of being correctly reported with accurate values independently of the presence of a valid application. Note that this shall also be interpreted to exclude dataIdentifiers used for identifying reprogrammable software components (e.g., F188H – Vehicle Manufacturer ECU Software Number) from being supported in the programmingSession as these software components may not be resident while the programmingSession is active.

If the ECU meets the definition of ECU Variant #1 (Programmable ECU via a bootloader) as defined in section 1.9.1, only the dataIdentifiers detailed in this specification shall be supported in the programmingSession. Exceptions are allowed but only when explicitly approved by Ford Core Network Communications.

Any non-volatile SWDL programmable area which has an associated file that is released with a WERS part number shall have this released part number embedded somewhere within the data section of the file. The number shall be readable in all diagnostic sessions other than the programmingSession. Note that this requirement does not apply to the secondary bootloader since the SBL does not remain resident on the ECU.

### dataIdentifier F111H (ECU Core Assembly Number)

This dataIdentifier is mandatory for ECU parts released through WERS and shall report all characters of the released WERS part number (including hyphens).

This number shall identify the combination of the ECU hardware (e.g., circuit board, micro-controller, memory, etc.) and any non-replaceable software (bootloaders and other fixed software).

The core assembly number will not represent:

* Any variable aspect of the ECU that is not associated with the circuit board (e.g., color or associated internal actuators or mechatronic parts) so long as these parts do not affect the compatibility of any downloadable component)
* Any downloadable software component regardless of whether or not his component is pre-loaded into the ECU.

Note that the core assembly number will provide sufficient information to fully define the compatibility but not necessarily the correct vehicle applicability of any downloadable software component.

Table 3.9 - ReadDataByIdentifier F111H request message flow

|  |  |  |  |
| --- | --- | --- | --- |
| Message direction: | | Tester → ECU | |
| Message Type: | | Request | |
| Data byte | Description (all values are in hexadecimal) | | Byte Value (Hex) |
| #1 | ReadDataByIdentifier request SID | | 22 |
| #2 | dataIdentifier [byte 1] (MSB) | | F1 |
| #3 | dataIdentifier [byte 2] (LSB) | | 11 |

Table 3.10 - ReadDataByIdentifier F111H positive response message flow

| Message direction: | | ECU → Tester | |
| --- | --- | --- | --- |
| Message Type: | | Response | |
| Data byte | Description (all values are in hexadecimal) | | Byte Value (Hex) |
| #1 | ReadDataByIdentifier response SID | | 62 |
| #2 | dataIdentifier [byte 1] (MSB) | | F1 |
| #3 | dataIdentifier [byte 2] (LSB) | | 11 |
| #4 | dataRecord [data\_1] | | ASCII |
| #5 | dataRecord [data\_2] | | ASCII |
| : | : | | : |
| #26 | dataRecord [data\_23] | | ASCII |
| #27 | dataRecord [data\_24] | | ASCII |

Number of bytes for ECU Core Assembly Number is twenty-four. The ECU Core Assembly Number data shall be encoded in ASCII, first character starts in dataRecord[data\_1], and all unused bytes shall be padded with 00H.

### dataIdentifier F113H (ECU Delivery Assembly Number)

This dataIdentifier is mandatory for ECU parts released through WERS and shall report all characters of the released WERS part number (including hyphens).

This number shall identify the complete ECU at the point of deliver to the assembly plant or the service bay. The number shall not be altered or updated after initial delivery of the ECU. The ECU Delivery Assembly Number shall represent the ECU Core Assembly Number F111H and additionally:

* Any variable aspect of the ECU (e.g., its color, associated internal actuators or mechatronic parts).
* All downloadable software components that are pre-loaded into the ECU.

Table 3.11 - ReadDataByIdentifier F113H request message flow

|  |  |  |  |
| --- | --- | --- | --- |
| Message direction: | | Tester → ECU | |
| Message Type: | | Request | |
| Data byte | Description (all values are in hexadecimal) | | Byte Value (Hex) |
| #1 | ReadDataByIdentifier request SID | | 22 |
| #2 | dataIdentifier [byte 1] (MSB) | | F1 |
| #3 | dataIdentifier [byte 2] (LSB) | | 13 |

Table 3.12 - ReadDataByIdentifier F113H positive response message flow

| Message direction: | | ECU → Tester | |
| --- | --- | --- | --- |
| Message Type: | | Response | |
| Data byte | Description (all values are in hexadecimal) | | Byte Value (Hex) |
| #1 | ReadDataByIdentifier response SID | | 62 |
| #2 | dataIdentifier [byte 1] (MSB) | | F1 |
| #3 | dataIdentifier [byte 2] (LSB) | | 13 |
| #4 | dataRecord [data\_1] | | ASCII |
| #5 | dataRecord [data\_2] | | ASCII |
| : | : | | : |
| #26 | dataRecord [data\_23] | | ASCII |
| #27 | dataRecord [data\_24] | | ASCII |

Number of bytes for ECU Delivery Assembly Number is twenty-four. The ECU Delivery Assembly Number data shall be encoded in ASCII, first character starts in dataRecord[data\_1], and all unused bytes shall be padded with 00H.

### dataIdentifier F18CH (ECU Serial Number)

This dataIdentifier shall be used to report a unique serial number for the ECU. Under special circumstances, the diagnostic feature owner may authorize the ECU serial number to be loaded with a string constant instead of a real unique serial number.

Table 3.13 – ReadDataByIdentifier F18CH request message flow

|  |  |  |  |
| --- | --- | --- | --- |
| Message direction: | | Tester → ECU | |
| Message Type: | | Request | |
| Data byte | Description (all values are in hexadecimal) | | Byte Value (Hex) |
| #1 | ReadDataByIdentifier request SID | | 22 |
| #2 | dataIdentifier [byte 1] (MSB) | | F1 |
| #3 | dataIdentifier [byte 2] (LSB) | | 8C |

Table 3.14 - ReadDataByIdentifier F18CH positive response message flow

| Message direction: | | ECU → Tester | |
| --- | --- | --- | --- |
| Message Type: | | Response | |
| Data byte | Description (all values are in hexadecimal) | | Byte Value (Hex) |
| #1 | ReadDataByIdentifier response SID | | 62 |
| #2 | dataIdentifier [byte 1] (MSB) | | F1 |
| #3 | dataIdentifier [byte 2] (LSB) | | 8C |
| #4 | dataRecord [data\_1] | | ASCII |
| #5 | dataRecord [data\_2] | | ASCII |
| : | : | | : |
| #18 | dataRecord [data\_15] | | ASCII |
| #19 | dataRecord [data\_16] | | ASCII |

Number of bytes for ECU Serial Number is sixteen. The ECU Serial Number data shall be encoded in ASCII, first character starts in dataRecord[data\_1], and all unused bytes shall be padded with 00H.

### dataIdentifier F180H (Boot Software Identification)

This dataIdentifier is mandatory for primary bootloader software released through WERS and shall in this case report all characters of the released WERS part number (including hyphens). This dataIdentifier shall not be supported if the primary bootloader software is not released through WERS.

Table 3.15 - ReadDataByIdentifier F180H request message flow

|  |  |  |  |
| --- | --- | --- | --- |
| Message direction: | | Tester → ECU | |
| Message Type: | | Request | |
| Data byte | Description (all values are in hexadecimal) | | Byte Value (Hex) |
| #1 | ReadDataByIdentifier request SID | | 22 |
| #2 | dataIdentifier [byte 1] (MSB) | | F1 |
| #3 | dataIdentifier [byte 2] (LSB) | | 80 |

Table 3.16 - ReadDataByIdentifier F180H positive response message flow

|  |  |  |  |
| --- | --- | --- | --- |
| Message direction: | | ECU → Tester | |
| Message Type: | | Response | |
| Data byte | Description (all values are in hexadecimal) | | Byte Value (Hex) |
| #1 | ReadDataByIdentifier response SID | | 62 |
| #2 | dataIdentifier [byte 1] (MSB) | | F1 |
| #3 | dataIdentifier [byte 2] (LSB) | | 80 |
| #4 | numberOfModules | | 01 |
| #5 | identificationParameterRecord [data\_1] (PBL SW part number, MSB) | | ASCII |
| #6 | identificationParameterRecord [data\_2] (PBL SW part number) | | ASCII |
| : | : | | : |
| #27 | identificationParameterRecord [data\_23] (PBL SW part number,) | | ASCII |
| #28 | identificationParameterRecord [data\_24] (PBL SW part number, LSB) | | ASCII |

The PBL SW part number (Boot Software Identification) data shall be encoded in ASCII, first character starts in identificationParameterRecord[data\_1], and all unused bytes shall be padded with 00H.

### dataIdentifier F109H (Boot Software Version Number)

This dataIdentifier is mandatory for primary bootloader software that does not support dataIdentifier F180H. In all other cases this dataIdentifier is optional. This dataIdentifier shall contain version information values specified by the supplier which allow a tool to uniquely identify the primary bootloader software resident on the ECU.

Table 3.17 - ReadDataByIdentifier F109H request message flow

|  |  |  |  |
| --- | --- | --- | --- |
| Message direction: | | Tester → ECU | |
| Message Type: | | Request | |
| Data byte | Description (all values are in hexadecimal) | | Byte Value (Hex) |
| #1 | ReadDataByIdentifier request SID | | 22 |
| #2 | dataIdentifier [byte 1] (MSB) | | F1 |
| #3 | dataIdentifier [byte 2] (LSB) | | 09 |

Table 3.18 - ReadDataByIdentifier F109H positive response message flow

| Message direction: | | ECU → Tester | |
| --- | --- | --- | --- |
| Message Type: | | Response | |
| Data byte | Description (all values are in hexadecimal) | | Byte Value (Hex) |
| #1 | ReadDataByIdentifier response SID | | 62 |
| #2 | dataIdentifier [byte 1] (MSB) | | F1 |
| #3 | dataIdentifier [byte 2] (LSB) | | 09 |
| #4 | Boot Software Version Number parameter 1 – Main Version | | BCD |
| #5 | Boot Software Version Number parameter 2 – Sub Version | | BCD |
| #6 | Boot Software Version Number parameter 3 – Release Version | | BCD |

### dataIdentifier D100H (Active Diagnostic Session)

The PBL can be in one of two different sessions, the defaultSession and the programmingSession. This dataIdentifier shall be used to report the active diagnostic session that the bootloader is in (refer to Figure 6.1) If the ECU has entered the programmingSession the first byte in the dataRecord shall have the value of 02H (programmingSession). If the ECU is in the defaultSession, the first byte in the dataRecord shall have the value of 01H (defaultSession). Note that the defaultSession within the primary bootloader is **not** the same as the defaultSession within the application. The primary bootloader defaultSession is only accessible upon boot up when the ECU has determined that no valid application is present (see section 6.1 for more details).

Table 3.19 - ReadDataByIdentifier D100H request message flow

|  |  |  |  |
| --- | --- | --- | --- |
| Message direction: | | Tester → ECU | |
| Message Type: | | Request | |
| Data byte | Description (all values are in hexadecimal) | | Byte Value (Hex) |
| #1 | ReadDataByIdentifier request SID | | 22 |
| #2 | dataIdentifier [byte 1] (MSB) | | D1 |
| #3 | dataIdentifier [byte 2] (LSB) | | 00 |

Table 3.20 - ReadDataByIdentifier D100H positive response message flow

| Message direction: | | ECU → Tester | |
| --- | --- | --- | --- |
| Message Type: | | Response | |
| Data byte | Description (all values are in hexadecimal) | | Byte Value (Hex) |
| #1 | ReadDataByIdentifier response SID | | 62 |
| #2 | dataIdentifier [byte 1] (MSB) | | D1 |
| #3 | dataIdentifier [byte 2] (LSB) | | 00 |
| #4 | dataRecord[data\_1] = defaultSession or programmingSession | | 01 or 02 |

### dataIdentifier F162H (Software Download Specification Version)

This dataIdentifier shall be used to report the software download specification an ECU is compliant with.

Table 3.21 - ReadDataByIdentifier F162H request message flow

|  |  |  |  |
| --- | --- | --- | --- |
| Message direction: | | Tester → ECU | |
| Message Type: | | Request | |
| Data byte | Description (all values are in hexadecimal) | | Byte Value (Hex) |
| #1 | ReadDataByIdentifier request SID | | 22 |
| #2 | dataIdentifier [byte 1] (MSB) | | F1 |
| #3 | dataIdentifier [byte 2] (LSB) | | 62 |

Table 3.22 - ReadDataByIdentifier F162H positive response message flow

| Message direction: | | ECU → Tester | |
| --- | --- | --- | --- |
| Message Type: | | Response | |
| Data byte | Description (all values are in hexadecimal) | | Byte Value (Hex) |
| #1 | ReadDataByIdentifier response SID | | 62 |
| #2 | dataIdentifier [byte 1] (MSB) | | F1 |
| #3 | dataIdentifier [byte 2] (LSB) | | 62 |
| #4 | dataRecord[data\_1] = 08 (SWDL 00.06.15.002-008) | | 08 |

### dataIdentifier D028H (Application Signature Validation Status)

This dataIdentifier shall be used to report current application signature status.

Table 3.23 - ReadDataByIdentifier D028H request message flow

|  |  |  |  |
| --- | --- | --- | --- |
| Message direction: | | Tester → ECU | |
| Message Type: | | Request | |
| Data byte | Description (all values are in hexadecimal) | | Byte Value (Hex) |
| #1 | ReadDataByIdentifier request SID | | 22 |
| #2 | dataIdentifier [byte 1] (MSB) | | D0 |
| #3 | dataIdentifier [byte 2] (LSB) | | 28 |

Table 3.24 - ReadDataByIdentifier D028H positive response message flow

| Message direction: | | ECU → Tester | |
| --- | --- | --- | --- |
| Message Type: | | Response | |
| Data byte | Description (all values are in hexadecimal) | | Byte Value (Hex) |
| #1 | ReadDataByIdentifier response SID | | 62 |
| #2 | dataIdentifier [byte 1] (MSB) | | D0 |
| #3 | dataIdentifier [byte 2] (LSB) | | 28 |
| #4 | dataRecord[data\_1] = application signature validation status | | State Encoded |

The state encoded values in dataRecord[data\_1] are defined in [Ford GMRDB].

### dataIdentifier D03FH (In-Use Application Signing Public Key Hash)

This dataIdentifier shall be used to report the hash of the in-use public key used for application code signing.

Table 3.25 - ReadDataByIdentifier D03FH request message flow

|  |  |  |  |
| --- | --- | --- | --- |
| Message direction: | | Tester → ECU | |
| Message Type: | | Request | |
| Data byte | Description (all values are in hexadecimal) | | Byte Value (Hex) |
| #1 | ReadDataByIdentifier request SID | | 22 |
| #2 | dataIdentifier [byte 1] (MSB) | | D0 |
| #3 | dataIdentifier [byte 2] (LSB) | | 3F |

Table 3.26 - ReadDataByIdentifier D03FH positive response message flow

| Message direction: | | ECU → Tester | |
| --- | --- | --- | --- |
| Message Type: | | Response | |
| Data byte | Description (all values are in hexadecimal) | | Byte Value (Hex) |
| #1 | ReadDataByIdentifier response SID | | 62 |
| #2 | dataIdentifier [byte 1] (MSB) | | D0 |
| #3 | dataIdentifier [byte 2] (LSB) | | 3F |
| #4 | dataRecord[data\_1] = Hash byte #1 (high byte) | | 00-FF |
| #5 | dataRecord[data\_2] = Hash byte #2 | | 00-FF |
| : | : | | : |
| #34 | dataRecord[data\_31] = Hash byte #31 | | 00-FF |
| #35 | dataRecord[data\_32] = Hash byte #32 (low byte) | | 00-FF |

### dataIdentifier F17FH (Ford Electronic Serial Number)

This dataIdentifier shall be used to report the Ford Electronic Serial Number.

Table 3.27 - ReadDataByIdentifier F17FH request message flow

|  |  |  |  |
| --- | --- | --- | --- |
| Message direction: | | Tester → ECU | |
| Message Type: | | Request | |
| Data byte | Description (all values are in hexadecimal) | | Byte Value (Hex) |
| #1 | ReadDataByIdentifier request SID | | 22 |
| #2 | dataIdentifier [byte 1] (MSB) | | F1 |
| #3 | dataIdentifier [byte 2] (LSB) | | 7F |

Table 3.28 - ReadDataByIdentifier F17FH positive response message flow

| Message direction: | | ECU → Tester | |
| --- | --- | --- | --- |
| Message Type: | | Response | |
| Data byte | Description (all values are in hexadecimal) | | Byte Value (Hex) |
| #1 | ReadDataByIdentifier response SID | | 62 |
| #2 | dataIdentifier [byte 1] (MSB) | | F1 |
| #3 | dataIdentifier [byte 2] (LSB) | | 7F |
| #4 | dataRecord [data\_1] | | ASCII |
| #5 | dataRecord [data\_2] | | ASCII |
| : | : | | : |
| #11 | dataRecord [data\_8] | | ASCII |

Number of bytes for Ford Electronic Serial Number is eight.

### Supported Negative Response Codes

See [ISO 14229-1] for ReadDataByIdentifier supported negative response codes.

## WriteDataByIdentifier (2EH) service

The WriteDataByIdentifier service allows the tester to write information into the ECU at an internal location logically specified by the provided dataIdentifier. The services shall be used to "write data" once to the ECU and the bootloader. The main user of this service is the ECU supplier, who can customize the ECU (e.g., core assembly part number and serial number). This service makes it possible to use the same primary bootloader in ECUs using the same microcontroller.

The support of WriteDataByIdentifier within the programmingSession is in general not supported by a vehicle manufacturer's test tools at the vehicle manufacturer's end of line or at service facilities. Unless specific agreements are made with all affected vehicle manufacturer's tools, an ECU shall be delivered to the vehicle manufacturer with the correct data already written to any supported dataIdentifiers which support the WriteDataByIdentifier service.

The WriteDataByIdentifier service normally writes data to flash or EEPROM. This means that the SBL needs to be downloaded before the writeDataByIdentifier service can be performed.

The WriteDataByIdentifier service is optional. ECU suppliers may implement other methods of populating the data for the ECU identification dataIdentifiers required by this specification. For example, although the data for dataIdentifier F113H may be writeable using service 2EH with the secondary bootloader, the data for dataIdentifier F111H may be hardcoded within the PBL itself, while the data for dataIdentifier F18CH may be writeable into EEPROM by the ECU's application and read from a fixed known memory location by the bootloader, etc. Regardless of the implementation, the ECU identification data must be correct upon delivery to Ford.

**NOTE**: The "write once" capability described for service 2EH within the programmingSession is intended to prevent the capability of an individual modifying this static part number information, etc., after delivery to Ford. However, as this information is required to be correct when the ECU is delivered to Ford, suppliers may need to have a supplier specific mechanism with security protections (e.g., supplier specific SBL) for updating dataIdentifier values that may have the necessity of being changed after being programmed by the supplier but prior to delivery to Ford. For example, it is possible that ECUs originally programmed with a given ECU Delivery Assembly Number value may require a software change prior to delivery to Ford, thus resulting in a new value required to be reported by the ECU Delivery Assembly Number DID when the ECU is actually delivered to Ford.

### dataIdentifier F111H (ECU Core Assembly Number)

Used for programming of the ECU Core Assembly Number. This service with dataIdentifier F111H is "write once" only. Support of this dataIdentifier with WriteDataByIdentifier is optional and if implemented shall be a number from the WERS numbering system.

Table 3.29 - WriteDataByIdentifier F111H request message flow

|  |  |  |  |
| --- | --- | --- | --- |
| Message direction: | | Tester → ECU | |
| Message Type: | | Request | |
| Data byte | Description (all values are in hexadecimal) | | Byte Value (Hex) |
| #1 | WriteDataByIdentifier request SID | | 2E |
| #2 | dataIdentifier [byte 1] (MSB) | | F1 |
| #3 | dataIdentifier [byte 2] (LSB) | | 11 |
| #4 | dataRecord [data\_1] | | ASCII |
| #5 | dataRecord [data\_2] | | ASCII |
| : | : | | : |
| #26 | dataRecord [data\_23] | | ASCII |
| #27 | dataRecord [data\_24] | | ASCII |

Number of bytes for ECU Core Assembly Number is twenty-four. The ECU Core Assembly Number data shall be encoded in ASCII, first character starts in dataRecord[data\_1], and all unused bytes shall be padded with 00H. The non-programmed value of the ECU Core Assembly Number shall be twenty-four bytes of FFH.

Table 3.30 - WriteDataByIdentifier F111H positive response message flow

| Message direction: | | ECU → Tester | |
| --- | --- | --- | --- |
| Message Type: | | Response | |
| Data byte | Description (all values are in hexadecimal) | | Byte Value (Hex) |
| #1 | WriteDataByIdentifier response SID | | 6E |
| #2 | dataIdentifier [byte 1] (MSB) | | F1 |
| #3 | dataIdentifier [byte 2] (LSB) | | 11 |

### dataIdentifier F113H (ECU Delivery Assembly Number)

Used for programming of the ECU Delivery Assembly Number. This service with dataIdentifier F113H is "write once" only. Support of this dataIdentifier with WriteDataByIdentifier is optional and if implemented shall be a number from the WERS numbering system.

Table 3.31 - WriteDataByIdentifier F113H request message flow

|  |  |  |  |
| --- | --- | --- | --- |
| Message direction: | | Tester → ECU | |
| Message Type: | | Request | |
| Data byte | Description (all values are in hexadecimal) | | Byte Value (Hex) |
| #1 | WriteDataByIdentifier request SID | | 2E |
| #2 | dataIdentifier [byte 1] (MSB) | | F1 |
| #3 | dataIdentifier [byte 2] (LSB) | | 13 |
| #4 | dataRecord [data\_1] | | ASCII |
| #5 | dataRecord [data\_2] | | ASCII |
| : | : | | : |
| #26 | dataRecord [data\_23] | | ASCII |
| #27 | dataRecord [data\_24] | | ASCII |

Number of bytes for ECU Delivery Assembly Number is twenty-four. The ECU Delivery Assembly Number data shall be encoded in ASCII, first character starts in dataRecord[data\_1], and all unused bytes shall be padded with 00H. The non-programmed value of the ECU Delivery Assembly Number shall be twenty-four bytes of FFH.

Table 3.32 - WriteDataByIdentifier F113H positive response message flow

| Message direction: | | ECU → Tester | |
| --- | --- | --- | --- |
| Message Type: | | Response | |
| Data byte | Description (all values are in hexadecimal) | | Byte Value (Hex) |
| #1 | WriteDataByIdentifier response SID | | 6E |
| #2 | dataIdentifier [byte 1] (MSB) | | F1 |
| #3 | dataIdentifier [byte 2] (LSB) | | 13 |

### dataIdentifier F18CH (ECU Serial Number)

Used for programming of the ECU Serial Number. This service with dataIdentifier F18CH is "write once" only. Support of this dataIdentifier with WriteDataByIdentifier is optional.

Table 3.33 - WriteDataByIdentifier F18CH request message flow

|  |  |  |  |
| --- | --- | --- | --- |
| Message direction: | | Tester → ECU | |
| Message Type: | | Request | |
| Data byte | Description (all values are in hexadecimal) | | Byte Value (Hex) |
| #1 | WriteDataByIdentifier request SID | | 2E |
| #2 | dataIdentifier [byte 1] (MSB) | | F1 |
| #3 | dataIdentifier [byte 2] (LSB) | | 8C |
| #4 | dataRecord [data\_1] | | ASCII |
| #5 | dataRecord [data\_2] | | ASCII |
| : | : | | : |
| #18 | dataRecord [data\_15] | | ASCII |
| #19 | dataRecord [data\_16] | | ASCII |

Number of bytes for ECU Serial Number is sixteen. The ECU Serial Number data shall be encoded in ASCII, first character starts in dataRecord[data\_1], and all unused bytes shall be padded with 00H.

The non-programmed value of ECU Serial Number shall be sixteen bytes of FFH.

Table 3.34 - WriteDataByIdentifier F18CH positive response message flow

| Message direction: | | ECU → Tester | |
| --- | --- | --- | --- |
| Message Type: | | Response | |
| Data byte | Description (all values are in hexadecimal) | | Byte Value (Hex) |
| #1 | WriteDataByIdentifier response SID | | 6E |
| #2 | dataIdentifier [byte 1] (MSB) | | F1 |
| #3 | dataIdentifier [byte 2] (LSB) | | 8C |

### dataIdentifier F17FH (Ford Electronic Serial Number)

Used for programming of the Ford Electronic Serial Number. This service with dataIdentifier F17FH is "write once" only. Support of this dataIdentifier with WriteDataByIdentifier is optional.

Table 3.35 - WriteDataByIdentifier F17FH request message flow

|  |  |  |  |
| --- | --- | --- | --- |
| Message direction: | | Tester → ECU | |
| Message Type: | | Request | |
| Data byte | Description (all values are in hexadecimal) | | Byte Value (Hex) |
| #1 | WriteDataByIdentifier request SID | | 2E |
| #2 | dataIdentifier [byte 1] (MSB) | | F1 |
| #3 | dataIdentifier [byte 2] (LSB) | | 7F |
| #4 | dataRecord [data\_1] | | ASCII |
| #5 | dataRecord [data\_2] | | ASCII |
| : | : | | : |
| #11 | dataRecord [data\_8] | | ASCII |

Number of bytes for ECU Serial Number is eight. The Ford Electronic Serial Number data shall be encoded in ASCII, first character starts in dataRecord[data\_1], and all unused bytes shall be padded with 00H.

The non-programmed value of Ford Electronic Serial Number shall be eight bytes of FFH.

Table 3.36 - WriteDataByIdentifier F17FH positive response message flow

| Message direction: | | ECU → Tester | |
| --- | --- | --- | --- |
| Message Type: | | Response | |
| Data byte | Description (all values are in hexadecimal) | | Byte Value (Hex) |
| #1 | WriteDataByIdentifier response SID | | 6E |
| #2 | dataIdentifier [byte 1] (MSB) | | F1 |
| #3 | dataIdentifier [byte 2] (LSB) | | 7F |

### Supported Negative Response Codes

See [ISO 14229-1] for WriteDataByIdentifier supported negative response codes.

If an ECU receives a validly formatted WriteDataByIdentifier request to a known supported "write once" dataIdentifier and that dataIdentifier has already been written once, the ECU shall respond using a negative response code of 31H (requestOutOfRange).

If an ECU receives a validly formatted WriteDataByIdentifier request to a known supported writeable dataIdentifier and the required secondary bootloader has not been downloaded in order to successfully perform the memory write, the ECU shall respond using a negative response code of 22H (conditionsNotCorrect).

## RoutineControl (31H) service

The RoutineControl service shall be used for entering explicit gateway state access (if supported), activating the SBL after download to RAM, erasing flash memory, calculating checksums, programming security bytes (if supported), and other functions as specified in the following sub-sections. Refer to section 0 for definitions of RoutineType and RoutineStatus.

### routineIdentifier 0300H (Gateway State Access)

routineIdentifier 0300H shall be used to enter/exit the explicit gateway state access. This routineIdentifier shall only be implemented if the ECU is an explicit diagnostic gateway ECU (i.e., this routineIdentifier does not apply to an ECU which is a transparent CAN diagnostic gateway only).

ECUs that implement routineIdentifier 0300H in the bootloader shall also implement routineIdentifier 0307H in the application.

Table 3.37 - RoutineControl 0300H request message flow, startRoutine

|  |  |  |  |
| --- | --- | --- | --- |
| Message direction: | | Tester → ECU | |
| Message Type: | | Request | |
| Data byte | Description (all values are in hexadecimal) | | Byte Value (Hex) |
| #1 | RoutineControl request SID | | 31 |
| #2 | routineControlType = startRoutine | | 01 or 81 |
| #3 | routineIdentifier [byte 1] (MSB) | | 03 |
| #4 | routineIdentifier [byte 2] (LSB) | | 00 |
| #5 | routineControlOptionRecord = [  SubNetworkAddress | | 00-FF |
| #6 | SubNodeAddress ] | | 00-FF |

Table 3.38 - RoutineControl 0300H request message flow, stopRoutine

|  |  |  |  |
| --- | --- | --- | --- |
| Message direction: | | Tester → ECU | |
| Message Type: | | Request | |
| Data byte | Description (all values are in hexadecimal) | | Byte Value (Hex) |
| #1 | RoutineControl request SID | | 31 |
| #2 | routineControlType = stopRoutine | | 02 or 82 |
| #3 | routineIdentifier [byte 1] (MSB) | | 03 |
| #4 | routineIdentifier [byte 2] (LSB) | | 00 |

Table 3.39 - RoutineControl 0300H request message flow, requestRoutineResults

|  |  |  |  |
| --- | --- | --- | --- |
| Message direction: | | Tester → ECU | |
| Message Type: | | Request | |
| Data byte | Description (all values are in hexadecimal) | | Byte Value (Hex) |
| #1 | RoutineControl request SID | | 31 |
| #2 | routineControlType = requestRoutineResults | | 03 or 83 |
| #3 | routineIdentifier [byte 1] (MSB) | | 03 |
| #4 | routineIdentifier [byte 2] (LSB) | | 00 |

Table 3.40 - RoutineControl 0300H positive response message flow, startRoutine

| Message direction: | | ECU → Tester | |
| --- | --- | --- | --- |
| Message Type: | | Response | |
| Data byte | Description (all values are in hexadecimal) | | Byte Value (Hex) |
| #1 | RoutineControl response SID | | 71 |
| #2 | routineControlType = startRoutine | | 01 |
| #3 | routineIdentifier [byte 1] (MSB) | | 03 |
| #4 | routineIdentifier [byte 2] (LSB) | | 00 |
| #5 | RoutineInfo = [  RoutineType =3  RoutineStatus =2 ] | | 32 |

Table 3.41 - RoutineControl 0300H positive response message flow, stopRoutine

| Message direction: | | ECU → Tester | |
| --- | --- | --- | --- |
| Message Type: | | Response | |
| Data byte | Description (all values are in hexadecimal) | | Byte Value (Hex) |
| #1 | RoutineControl response SID | | 71 |
| #2 | routineControlType = stopRoutine | | 02 |
| #3 | routineIdentifier [byte 1] (MSB) | | 03 |
| #4 | routineIdentifier [byte 2] (LSB) | | 00 |
| #5 | RoutineInfo = [  RoutineType = 3  RoutineStatus = 0 ] | | 30 |

Table 3.42 - RoutineControl 0300H positive response message flow, requestRoutineResults

| Message direction: | | ECU → Tester | |
| --- | --- | --- | --- |
| Message Type: | | Response | |
| Data byte | Description (all values are in hexadecimal) | | Byte Value (Hex) |
| #1 | RoutineControl response SID | | 71 |
| #2 | routineControlType = requestRoutineResults | | 03 |
| #3 | routineIdentifier [byte 1] (MSB) | | 03 |
| #4 | routineIdentifier [byte 2] (LSB) | | 00 |
| #5 | RoutineInfo = [  RoutineType = 3  RoutineStatus = 0, 1 or 2 ] | | 30 or 31 or 32 |

### routineIdentifier 0301H (Activate Secondary Bootloader)

routineIdentifier 0301H shall be used to activate the SBL after download to RAM.

Table 3.43 - RoutineControl 0301H request message flow

|  |  |  |  |
| --- | --- | --- | --- |
| Message direction: | | Tester → ECU | |
| Message Type: | | Request | |
| Data byte | Description (all values are in hexadecimal) | | Byte Value (Hex) |
| #1 | RoutineControl request SID | | 31 |
| #2 | routineControlType = startRoutine | | 01 or 81 |
| #3 | routineIdentifier [byte 1] (MSB) | | 03 |
| #4 | routineIdentifier [byte 2] (LSB) | | 01 |
| #5 | routineControlOptionRecord = [  memoryAddress (SBL start address, byte 1, MSB) | | 00-FF |
| #6 | memoryAddress (SBL start address, byte 2) | | 00-FF |
| #7 | memoryAddress (SBL start address, byte 3) | | 00-FF |
| #8 | memoryAddress (SBL start address, byte 4, LSB) ] | | 00-FF |

Table 3.44 - RoutineControl 0301H positive response message flow

| Message direction: | | ECU → Tester | |
| --- | --- | --- | --- |
| Message Type: | | Response | |
| Data byte | Description (all values are in hexadecimal) | | Byte Value (Hex) |
| #1 | RoutineControl response SID | | 71 |
| #2 | routineControlType = startRoutine | | 01 |
| #3 | routineIdentifier [byte 1] (MSB) | | 03 |
| #4 | routineIdentifier [byte 2] (LSB) | | 01 |
| #5 | RoutineInfo = [  RoutineType 1  RoutineStatus 0 ] | | 10 |

### routineIdentifier FF00H (eraseMemory)

routineIdentifier FF00H shall be used to perform a flash memory erase.

If an ECU is delivered to the OEM with the complete programmable memory already erased, the ECU shall include an internal "already erased" detection in order to reduce the overall software download time. If the complete programmable memory already is erased and all other preconditions are met (e.g., SBL is downloaded and activated), the ECU shall send a positive response to a RoutineControl eraseMemory request within P2Server timing. Once the ECU has programmed any bytes within the programmable memory area, the internal "already erased" detection shall be blocked and an erase operation shall always be performed.

Table 3.45 - RoutineControl FF00H request message flow

|  |  |  |  |
| --- | --- | --- | --- |
| Message direction: | | Tester → ECU | |
| Message Type: | | Request | |
| Data byte | Description (all values are in hexadecimal) | | Byte Value (Hex) |
| #1 | RoutineControl request SID | | 31 |
| #2 | routineControlType = startRoutine | | 01 or 81 |
| #3 | routineIdentifier [byte 1] (MSB) | | FF |
| #4 | routineIdentifier [byte 2] (LSB) | | 00 |
| #5 | routineControlOptionRecord = [  memoryAddress (Erase start address, byte 1, MSB) | | 00-FF |
| #6 | memoryAddress (Erase start address, byte 2) | | 00-FF |
| #7 | memoryAddress (Erase start address, byte 3) | | 00-FF |
| #8 | memoryAddress (Erase start address, byte 4, LSB) | | 00-FF |
| #9 | memorySize (Erase length, byte 1, MSB) | | 00-FF |
| #10 | memorySize (Erase length, byte 2) | | 00-FF |
| #11 | memorySize (Erase length, byte 3) | | 00-FF |
| #12 | memorySize (Erase length, byte 4, LSB) ] | | 00-FF |

Table 3.46 - RoutineControl FF00H positive response message flow

| Message direction: | | ECU → Tester | |
| --- | --- | --- | --- |
| Message Type: | | Response | |
| Data byte | Description (all values are in hexadecimal) | | Byte Value (Hex) |
| #1 | RoutineControl response SID | | 71 |
| #2 | routineControlType = startRoutine | | 01 |
| #3 | routineIdentifier [byte 1] (MSB) | | FF |
| #4 | routineIdentifier [byte 2] (LSB) | | 00 |
| #5 | RoutineInfo = [  RoutineType 1  RoutineStatus 0 ] | | 10 |

### routineIdentifier FF01H (checkProgrammingDependencies)

routineIdentifier FF01H shall be used to initiate a checksum calculation of an ECU memory block. The ECU memory block requested by a tester can include any portion of memory that contains downloadable software components that are pre-loaded into the ECU and can contain any portion of memory that contains non-replaceable software (e.g., bootloaders, boot managers, and other fixed software). Implementations shall restrict using this routine for memory sizes less than 128 bytes. RoutineControl with routine identifier FF01H is not used during a normal software download or upload operation but can be used for debugging. The same checksum algorithm as used by the RequestTransferExit service shall be used.

Table 3.47 - RoutineControl FF01H request message flow

|  |  |  |  |
| --- | --- | --- | --- |
| Message direction: | | Tester → ECU | |
| Message Type: | | Request | |
| Data byte | Description (all values are in hexadecimal) | | Byte Value (Hex) |
| #1 | RoutineControl request SID | | 31 |
| #2 | routineControlType = startRoutine | | 01 or 81 |
| #3 | routineIdentifier [byte 1] (MSB) | | FF |
| #4 | routineIdentifier [byte 2] (LSB) | | 01 |
| #5 | routineControlOptionRecord = [  memoryAddress (Checksum start address, byte 1, MSB) | | 00-FF |
| #6 | memoryAddress (Checksum start, byte 2) | | 00-FF |
| #7 | memoryAddress (Checksum start, byte 3) | | 00-FF |
| #8 | memoryAddress (Checksum start, byte 4, LSB) | | 00-FF |
| #9 | memorySize (Length, byte 1, MSB) | | 00-FF |
| #10 | memorySize (Length, byte 2) | | 00-FF |
| #11 | memorySize (Length, byte 3) | | 00-FF |
| #12 | memorySize (Length, byte 4, LSB) ] | | 00-FF |

Table 3.48 - RoutineControl FF01H positive response message flow

| Message direction: | | ECU → Tester | |
| --- | --- | --- | --- |
| Message Type: | | Response | |
| Data byte | Description (all values are in hexadecimal) | | Byte Value (Hex) |
| #1 | RoutineControl response SID | | 71 |
| #2 | routineControlType = startRoutine | | 01 |
| #3 | routineIdentifier [byte 1] (MSB) | | FF |
| #4 | routineIdentifier [byte 2] (LSB) | | 01 |
| #5 | RoutineInfo = [  RoutineType 1  RoutineStatus 0 ] | | 10 |
| #6 | routineStatusRecord [byte 1] = Checksum byte 1 (MSB) | | 00-FF |
| #7 | routineStatusRecord [byte 2] = Checksum byte 2 (LSB) | | 00-FF |

### routineIdentifier 0304H (Check Valid Application)

routineIdentifier 0304H shall be used to detect whether or not the ECU has valid application software programmed (see section 6.2 for more details) and report this determination to the tester.

Table 3.49 - RoutineControl 0304H request message flow

|  |  |  |  |
| --- | --- | --- | --- |
| Message direction: | | Tester → ECU | |
| Message Type: | | Request | |
| Data byte | Description (all values are in hexadecimal) | | Byte Value (Hex) |
| #1 | RoutineControl request SID | | 31 |
| #2 | routineControlType = startRoutine | | 01 or 81 |
| #3 | routineIdentifier [byte 1] (MSB) | | 03 |
| #4 | routineIdentifier [byte 2] (LSB) | | 04 |

Table 3.50 - RoutineControl 0304H positive response message flow

| Message direction: | | ECU → Tester | |
| --- | --- | --- | --- |
| Message Type: | | Response | |
| Data byte | Description (all values are in hexadecimal) | | Byte Value (Hex) |
| #1 | RoutineControl response SID | | 71 |
| #2 | routineControlType = startRoutine | | 01 |
| #3 | routineIdentifier [byte 1] (MSB) | | 03 |
| #4 | routineIdentifier [byte 2] (LSB) | | 04 |
| #5 | RoutineInfo = [  RoutineType 1  RoutineStatus 0 ] | | 10 |
| #6 | routineStatusRecord[byte 1] = ApplicationStatus = [  Valid Application Software is NOT Present  OR  Valid Application Software Is Present] | | 01  02 |

### routineIdentifier 0305H (Update Security Bytes)

Support of routineIdentifier 0305H is optional and when supported shall be used to write the fixed security byte information for security level 01H in order to customize the security algorithm in [Ford DPSA].

The non-programmed value for the fixed security byte information shall always consist of all bytes filled with FFH. An ECU shall reject a request to update the SWDL security bytes with NRC 31H if the security byte information has already been written (i.e., contains a value other than all FFH).

In general, fixed security byte information is present when an ECU is delivered to Ford. If the fixed security byte information is not delivered already programmed and therefore is required to be written by Ford test tools at Ford end of line or Ford service facilities, then explicit and specific agreements must be made with all affected Ford tools and facilities.

Table 3.51 - RoutineControl 0305H request message flow

|  |  |  |  |
| --- | --- | --- | --- |
| Message direction: | | Tester → ECU | |
| Message Type: | | Request | |
| Data byte | Description (all values are in hexadecimal) | | Byte Value (Hex) |
| #1 | RoutineControl request SID | | 31 |
| #2 | routineControlType = startRoutine | | 01 or 81 |
| #3 | routineIdentifier [byte 1] (MSB) | | 03 |
| #4 | routineIdentifier [byte 2] (LSB) | | 05 |
| #5 | routineControlOptionRecord = [  Security Level | | 01 |
| #6 | FixedByte #1 (high byte) | | 00-FF |
| : | : | | : |
| #n | FixedByte #m (low byte) | | 00-FF |

Table 3.52 - RoutineControl 0305H positive response message flow

| Message direction: | | ECU → Tester | |
| --- | --- | --- | --- |
| Message Type: | | Response | |
| Data byte | Description (all values are in hexadecimal) | | Byte Value (Hex) |
| #1 | RoutineControl response SID | | 71 |
| #2 | routineControlType = startRoutine | | 01 |
| #3 | routineIdentifier [byte 1] (MSB) | | 03 |
| #4 | routineIdentifier [byte 2] (LSB) | | 05 |
| #5 | RoutineInfo = [  RoutineType 1  RoutineStatus 0 ] | | 10 |

### routineIdentifier 021BH (Set OTA Update State)

routineIdentifier 021BH shall be used to set the OTA Update State of the bootloader. The bootloader may be in any of the following 3 OTA update states.

* Active partition update state
  + This is the normal and default state the bootloader is in whenever it is entered
* Inactive partition update state
  + This state is intended to allow updates to the inactive memory partition. When this state is active, all subsequent software download services that act on memory including erase memory, transfer data, calculate Swash, etc. shall be performed on the inactive memory partition for an ECU. The contents of the inactive memory shall become the contents of the active memory upon execution of routineIdentifier 021CH (refer to section 3.6.8).
* Rollback state
  + This state is intended to support a rollback to the previously active software following an OTA update of an inactive partition. When this state is active, the ability to update memory via a RequestDownload 34H or eraseMemory routine FF00H shall be blocked and responded to with a negative response code of 22H (conditionsNotCorrect). Depending on the inactive memory architecture type (see Annex F) the previously active software may be the same as the inactive partition or it may be separate.

This routineIdentifier shall only be supported prior to activation of the secondary bootloader. Once the secondary bootloader is activated, the current OTA Update State will be latched for the current duration of the programmingSession. If the secondary bootloader is already activated when this request is received, the ECU shall respond to a request for this routineIdentifier using a negative response code of 22H (conditionsNotCorrect).

Table 3.53 - RoutineControl 021BH request message flow

|  |  |  |  |
| --- | --- | --- | --- |
| Message direction: | | Tester → ECU | |
| Message Type: | | Request | |
| Data byte | Description (all values are in hexadecimal) | | Byte Value (Hex) |
| #1 | RoutineControl request SID | | 31 |
| #2 | routineControlType = startRoutine | | 01 or 81 |
| #3 | routineIdentifier [byte 1] (MSB) | | 02 |
| #4 | routineIdentifier [byte 2] (LSB) | | 1B |
| #5 | routineControlOptionRecord = [  OTA Update State  00 = Active partition update state  01 = Inactive partition update state  02 = Rollback state ] | | 00 - 02 |

Table 3.54 - RoutineControl 021BH positive response message flow

|  |  |  |  |
| --- | --- | --- | --- |
| Message direction: | | ECU → Tester | |
| Message Type: | | Response | |
| Data byte | Description (all values are in hexadecimal) | | Byte Value (Hex) |
| #1 | RoutineControl response SID | | 71 |
| #2 | routineControlType = startRoutine | | 01 |
| #3 | routineIdentifier [byte 1] (MSB) | | 02 |
| #4 | routineIdentifier [byte 2] (LSB) | | 1B |
| #5 | RoutineInfo = [  RoutineType 1  RoutineStatus 0 ] | | 10 |

### routineIdentifier 021AH (Prepare Inactive Memory)

routineIdentifier 021AH shall perform remaining operations needed to prepare the activation of 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.

In addition to security access (refer to section 3.3), this routineIdentifier shall have the following pre-conditions:

* The bootloader shall be in an OTA update state equal to “Inactive partition update state” as enabled by routineIdentifier 021BH Set OTA Update State (refer to 3.6.7)
* Secondary bootloader shall be currently activated

If these pre-conditions are not met then the ECU shall respond using a negative response code of 22H (conditionsNotCorrect).

If the ECU architecture supports an active back up (e.g., A/B/A type architecture), upon receiving this routine the ECU shall verify the active application and the OTA active backup 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).

In case of an X out of Y download of logical blocks (i.e., not all logical blocks have been updated in the inactive partition since the last successful swap), then this routine shall perform the following actions. For all logical blocks in the inactive partition not downloaded and validated since the last swap, if the logical block is not valid or if the logical block is valid but the root hash does not match the corresponding logical block in the active partition, then the ECU shall:

1. Erase the corresponding logical blocks in the inactive memory
2. Copy the logical blocks from the active partition to the inactive partition

Table 3.55 - RoutineControl 021AH request message flow

|  |  |  |  |
| --- | --- | --- | --- |
| Message direction: | | Tester → ECU | |
| Message Type: | | Request | |
| Data byte | Description (all values are in hexadecimal) | | Byte Value (Hex) |
| #1 | RoutineControl request SID | | 31 |
| #2 | routineControlType = startRoutine | | 01 or 81 |
| #3 | routineIdentifier [byte 1] (MSB) | | 02 |
| #4 | routineIdentifier [byte 2] (LSB) | | 1A |

Table 3.56 - RoutineControl 021AH positive response message flow

|  |  |  |  |
| --- | --- | --- | --- |
| Message direction: | | ECU → Tester | |
| Message Type: | | Response | |
| Data byte | Description (all values are in hexadecimal) | | Byte Value (Hex) |
| #1 | RoutineControl response SID | | 71 |
| #2 | routineControlType = startRoutine | | 01 |
| #3 | routineIdentifier [byte 1] (MSB) | | 02 |
| #4 | routineIdentifier [byte 2] (LSB) | | 1A |
| #5 | RoutineInfo = [  RoutineType 1  RoutineStatus 0 ] | | 10 |

### routineIdentifier 0219H (Calculate SWash)

routineIdentifier 0219H shall perform a hash of root hashes for the logical blocks associated with verification structure addresses included in the request.

If the current OTA update state is “Active partition update state” (see 3.6.7), this calculation shall be performed over the active partition. If the current OTA update state is “Inactive partition update state” (see 3.6.7), this calculation shall be performed over the inactive partition. If the current OTA update state is “Rollback state” (see 3.6.7), this calculation shall be performed over the active backup For ECUs that do not support an active backup (e.g., A/B implementations), if the current OTA update state is “Rollback state” this calculation shall be performed over the inactive partition.

The SWash may be used as a mechanism to verify a set of software matches what is expected by a client. The SWash calculation shall use the order of the verification structure addresses as specified below.



Figure 3.1 SWash Calculation

SWash calculation steps:

* + - 1. Compute root hash associated with each verification structure per [Ford ASRS].
      2. Align and combine root hash values in ascending numerical order of verification structure address
      3. Perform SHA256 hash of the output of step 2.

For example:

Target ECU receives a request with three verification structure addresses with associated RootHashes

VSA - RootHash

0x801FFF00 - 0xCF6822974AA52F6E596B81EB366529AA19B270CB6F615F85BA11FBC9362218D6

0x803FFF00 - 0x7648A086A5FA30B4F62FF44CADD7B90D3F70952024DFCD9A50D7AE44846F17BB

0x805FFF00 - 0xB4B55A0087DFCB59F99CE42E4C92E9EF111421DA2ED6FA3395996B872D4990B9

SWash = Sha256

(CF6822974AA52F6E596B81EB366529AA19B270CB6F615F85BA11FBC9362218D67648A086A5FA30B4F62FF44CADD7B90D3F70952024DFCD9A50D7AE44846F17BBB4B55A0087DFCB59F99CE42E4C92E9EF111421DA2ED6FA3395996B872D4990B9)

SWash = 0xEC43A131154FA4B635A420D7D5A634B300F89529272EE765A79CECF05D36A54B

If this routine request has a valid length, but a verification structure address included in the request is not associated with a logical block, the ECU shall respond using a negative response code of 31H (requestOutOfRange).

If this routine request has a valid length, and all verification structure addresses included in the request are associated with a logical block, but one or more logical blocks are not valid, the ECU shall respond using a negative response code of 72H (generalProgrammingError).

Table 3.57 - RoutineControl 0219H request message flow

|  |  |  |  |
| --- | --- | --- | --- |
| Message direction: | | Tester → ECU | |
| Message Type: | | Request | |
| Data byte | Description (all values are in hexadecimal) | | Byte Value (Hex) |
| #1 | RoutineControl request SID | | 31 |
| #2 | routineControlType = startRoutine | | 01 or 81 |
| #3 | routineIdentifier [byte 1] (MSB) | | 02 |
| #4 | routineIdentifier [byte 2] (LSB) | | 19 |
| #5  #6  #7  #8 | routineControlOptionRecord = [  verificationStructureAddress#1[] = [  Byte #1 (MSB)  Byte #2  Byte #3  Byte #4 (LSB) ] | | 00-FF  00-FF  00-FF  00-FF |
| : | : | | : |
| #(k\*4+1)  #(k\*4+2)  #(k\*4+3)  #(k\*4+4) | verificationStructureAddress#k[] = [  Byte #1 (MSB)  Byte #2  Byte #3  Byte #4 (LSB) ] | | 00-FF  00-FF  00-FF  00-FF |

Table 3.58 - RoutineControl 0219H positive response message flow

|  |  |  |  |
| --- | --- | --- | --- |
| Message direction: | | ECU → Tester | |
| Message Type: | | Response | |
| Data byte | Description (all values are in hexadecimal) | | Byte Value (Hex) |
| #1 | RoutineControl response SID | | 71 |
| #2 | routineControlType = startRoutine | | 01 |
| #3 | routineIdentifier [byte 1] (MSB) | | 02 |
| #4 | routineIdentifier [byte 2] (LSB) | | 19 |
| #5 | RoutineInfo = [  RoutineType 1  RoutineStatus 0 ] | | 10 |
| #6  :  #37 | routineStatusRecord = [  SWash#1 (MSB)  :  SWash#32 (LSB) ] | | 00-FF  00-FF |

### routineIdentifier 021CH (Replace Active Memory Partition)

This routine is used to perform any necessary actions for the ECU to begin execution of new software from either the inactive partition or the active backup following an exit of the programmingSession. Depending on the inactive memory architecture type (see Annex F) the ECU may not have a separate active backup and therefore the inactive partition contains the active backup after a swap. On completion of this routine, a tester will typically issue a functionally addressed ECU Reset to synchronously reboot ECUs. ECUs must be prepared to execute newly downloaded software without added delay to prevent systemic communication faults.

In addition to security access (refer to section 3.3), this routineIdentifier shall have the following pre-conditions:

* The bootloader shall be in an OTA update state equal to “Inactive partition update state” or “Rollback state” as enabled by routineIdentifier 021BH Set OTA Update State (refer to 3.6.7)
* Secondary bootloader shall be currently activated
* If OTA update state is equal to “Inactive partition update state”, then routineIdentifier 0304H (Check Valid Application) shall have been executed on the inactive memory partition and returned an ApplicationStatus of “Valid Application Software Is Present”. If any action has been taken that invalidates the application after this routineIdentifier was executed then this pre-condition is no longer met.
* routineIdentifier 021CH shall not have been previously successfully executed during the current programmingSession.

If none of these pre-conditions are met then the ECU shall respond using a negative response code of 22H (conditionsNotCorrect).

When OTA update state is equal to “Inactive partition update state”, routineIdentifier 021CH shall cause the contents of a fully programmed inactive memory partition to become the contents of the active memory partition. When OTA update state is equal to “Rollback state”, routineIdentifier 021CH shall cause the contents of the active backup to become the contents of the active memory partition. In case of A/B implementations (which do not support a separate active backup), when OTA update state is equal to “Rollback state”, routineIdentifier 021CH shall cause the contents of a fully programmed inactive memory partition to become the contents of the active memory partition.

The routine shall prepare any required memory swaps before returning the positive response. In case of an activation failure when OTA update state is equal to “Inactive partition update state”, the ECU shall automatically attempt to restore the active backup (refer to Annex F for implementation details).

The routine shall not reset the ECU.

Table 3.59 - RoutineControl 021CH request message flow

|  |  |  |  |
| --- | --- | --- | --- |
| Message direction: | | Tester → ECU | |
| Message Type: | | Request | |
| Data byte | Description (all values are in hexadecimal) | | Byte Value (Hex) |
| #1 | RoutineControl request SID | | 31 |
| #2 | routineControlType = startRoutine | | 01 or 81 |
| #3 | routineIdentifier [byte 1] (MSB) | | 02 |
| #4 | routineIdentifier [byte 2] (LSB) | | 1C |

Table 3.60 - RoutineControl 021CH positive response message flow

| Message direction: | | ECU → Tester | |
| --- | --- | --- | --- |
| Message Type: | | Response | |
| Data byte | Description (all values are in hexadecimal) | | Byte Value (Hex) |
| #1 | RoutineControl response SID | | 71 |
| #2 | routineControlType = startRoutine | | 01 |
| #3 | routineIdentifier [byte 1] (MSB) | | 02 |
| #4 | routineIdentifier [byte 2] (LSB) | | 1C |
| #5 | RoutineInfo = [  RoutineType 1  RoutineStatus 0 ] | | 10 |

### RoutineInfo

The RoutineInfo parameter is mandatory and shall always be present as the fifth byte in every service 31H positive response. RoutineInfo consists of two four bit parameters. The upper nibble (bits 7-4) shall be the RoutineType and the lower nibble (bits 3-0) shall be the RoutineStatus. Valid RoutineType values are defined in Table 3.61 and valid RoutineStatus values are defined in Table 3.62.

Table 3.61 – RoutineType

|  |  |
| --- | --- |
| RoutineType | |
| Hex | Definition |
| 0 | Reserved by this document |
| 1 | A positive response to sub-function 01H (startRoutine) shall only be given after the routine has stopped executing. Any Response Additional Data is returned in the positive response to the startRoutine request. |
| 2 | The routine shall start prior to the positive response to sub-function 01H (startRoutine). The routine shall complete after it has run for a finite time (i.e., it is not dependent upon a stopRoutine request sub-function 02H). Any request for routine results using sub-function 03H that is received while the routine is executing shall result in a negative response code of 21H (busyRepeatRequest). See Note 1. |
| 3 | The routine shall start prior to the positive response to sub-function 01H (startRoutine). The routine may complete after it has run for a finite time or it may run until commanded to stop via sub-function 02H (stopRoutine). Any request for routine results using sub-function 03H that is received while the routine is executing shall result in a positive response with the current results. See Note 1. |
| 4 - F | Reserved by this document |

Note 1: The results from this RoutineType shall remain accessible using requestRoutineResults so long as the diagnostic session during which the test was run remains active or until the next control routine is executed.

Table 3.62 – RoutineStatus

| RoutineStatus | |
| --- | --- |
| Hex | Definition |
| 0 | The routine completed all requested functionality |
| 1 | The routine aborted before completion (e.g., all documented fault monitoring was not executed during self-test because it was blocked by the presence of an active fault). |
| 2 | The routine is currently active |
| 3 - F | Reserved by this document |

### Supported Negative Response Codes

See [ISO 14229-1] RoutineControl supported negative response codes.

If an ECU receives a validly formatted RoutineControl request to a known supported routineIdentifier and the required secondary bootloader has not been downloaded in order to successfully perform the routine, the ECU shall respond using a negative response code of 22H (conditionsNotCorrect).

## RequestDownload (34H) service

The RequestDownload service is used by the tester to initiate a data transfer from the tester to the ECU (download).

The RequestDownload service includes a dataFormatIdentifier that specifies if the data is compressed and/or encrypted. The file to be downloaded shall contain information regarding how the data is represented in the file and therefore the value to use for the dataFormatIdentifier. If the file to download does not indicate encryption or compression of the data, a value of 00H shall be used for the dataFormatIdentifer. A dataFormatIdentifier value of 00H means the data sent over the network shall be written to ECU memory exactly as transmitted over the network. For any non-zero dataFormatIdentifier values (see [Ford DCES] for possible values), the data being transmitted over the network is encrypted and/or compressed (as compared to what will be stored in the ECU's memory) and therefore the ECU must uncompress and/or decrypt the data prior to writing it to memory. ECUs supporting the RequestDownload service are required to support the dataFormatIdentifier value of 00H.

Table 3.63 - RequestDownload request message flow

|  |  |  |  |
| --- | --- | --- | --- |
| Message direction: | | Tester → ECU | |
| Message Type: | | Request | |
| Data byte | Description (all values are in hexadecimal) | | Byte Value (Hex) |
| #1 | RequestDownload request SID | | 34 |
| #2 | dataFormatIdentifier | | 00-FF |
| #3 | addressAndLengthFormatIdentifier | | 44 |
| #4 | memoryAddress[Byte1] (MSB) | | 00-FF |
| #5 | memoryAddress[Byte2] | | 00-FF |
| #6 | memoryAddress[Byte3] | | 00-FF |
| #7 | memoryAddress[Byte4] (LSB) | | 00-FF |
| #8 | memorySize[Byte1] (MSB) | | 00-FF |
| #9 | memorySize[Byte2] | | 00-FF |
| #10 | memorySize[Byte3] | | 00-FF |
| #11 | memorySize[Byte4] (LSB) | | 00-FF |

### RequestDownload positive response

Table 3.64 - RequestDownload positive response message flow

| Message direction: | | ECU → Tester | |
| --- | --- | --- | --- |
| Message Type: | | Response | |
| Data byte | Description (all values are in hexadecimal) | | Byte Value (Hex) |
| #1 | RequestDownload response SID | | 74 |
| #2 | lengthFormatIdentifier | | 20 |
| #3 | maxNumberOfBlockLength [byte #1] | | 00-0F |
| #4 | maxNumberOfBlockLength [byte #2] | | 00-FF |

The maxNumberOfBlockLength shall be dimensioned to be as big as possible (i.e., all available RAM space shall be used). The absolute minimum allowed maxNumberOfBlockLength shall be 128 bytes unless explicit approval is granted by Ford Core Network Communications.

### Supported Negative Response Codes

See [ISO 14229-1] for RequestDownload supported negative response codes.

## RequestUpload (35H) service

The RequestUpload service is used by the tester to initiate a data transfer from the ECU to the tester (upload).

The RequestUpload service is optional.

Table 3.65 - RequestUpload request message flow

|  |  |  |  |
| --- | --- | --- | --- |
| Message direction: | | Tester → ECU | |
| Message Type: | | Request | |
| Data byte | Description (all values are in hexadecimal) | | Byte Value (Hex) |
| #1 | RequestUpload request SID | | 35 |
| #2 | dataFormatIdentifier | | 00 |
| #3 | addressAndLengthFormatIdentifier | | 44 |
| #4 | memoryAddress[Byte1] (MSB) | | 00-FF |
| #5 | memoryAddress[Byte2] | | 00-FF |
| #6 | memoryAddress[Byte3] | | 00-FF |
| #7 | memoryAddress[Byte4] (LSB) | | 00-FF |
| #8 | memorySize[Byte1] (MSB) | | 00-FF |
| #9 | memorySize[Byte2] | | 00-FF |
| #10 | memorySize[Byte3] | | 00-FF |
| #11 | memorySize[Byte4] (LSB) | | 00-FF |

### RequestUpload positive response

Table 3.66 - RequestUpload positive response message flow

| Message direction: | | ECU → Tester | |
| --- | --- | --- | --- |
| Message Type: | | Response | |
| **Data byte** | **Description (all values are in hexadecimal)** | | **Byte Value (Hex)** |
| #1 | RequestUpload response SID | | 75 |
| #2 | lengthFormatIdentifier | | 20 |
| #3 | maxNumberOfBlockLength [byte #1] | | 00-0F |
| #4 | maxNumberOfBlockLength [byte #2] | | 00-FF |

### Supported Negative Response Codes

See [ISO 14229-1] for RequestUpload supported negative response codes.

## TransferData (36H) service

The TransferData service is used by the tester to transfer data either from the tester to the ECU (download) or from the ECU to the tester (upload). The data transfer direction is defined by the preceding RequestDownload or RequestUpload service. If the tester initiated a RequestDownload the data to be transferred is included in the parameter(s) transferRequestParameter in the TransferData request message(s). If the tester initiated a RequestUpload the data to be uploaded is included in the parameter(s) transferResponseParameter in the TransferData response message(s)

The TransferData service request includes a blockSequenceCounter to allow for an improved error handling in case a TransferData service fails during a sequence of multiple TransferData requests. The four example use cases for blockSequenceCounter, specified in [ISO 14229-1], shall be handled by the tester and the ECU. It is recommended a tester retransmit the transferData service (i.e., with the same blockSequenceCounter and data) at least two additional times if no response is received from the ECU. If an ECU receives a transferData request during an active download sequence with the same blockSequenceCounter as the last accepted transferData request, it shall respond with a positive response without writing the data once again to its memory.

The blockSequenceCounter of the ECU shall be initialized to one (1) when receiving a RequestDownload (34H) or RequestUpload (35H) request message. This means that the first TransferData (36H) request message following the RequestDownload (34H) or RequestUpload (35H) request message starts with a blockSequenceCounter of one (1).

Table 3.67 - TransferData request message flow (download)

|  |  |  |  |
| --- | --- | --- | --- |
| Message direction: | | Tester → ECU (download) | |
| Message Type: | | Request | |
| Data byte | Description (all values are in hexadecimal) | | Byte Value (Hex) |
| #1 | TransferData request SID | | 36 |
| #2 | blockSequenceCounter | | 00-FF |
| #3 | transferRequestParameterRecord[transferRequestParameter#1] =  data#1 | | 00-FF |
| #4 | transferRequestParameterRecord[transferRequestParameter#2] =  data#2 | | 00-FF |
| #5 | transferRequestParameterRecord[transferRequestParameter#3] =  data#3 | | 00-FF |
| : | : | | : |
| #n | transferRequestParameterRecord[transferRequestParameter#n] =  data#n | | 00-FF |

Table 3.68 - TransferData request message flow (upload)

| Message direction: | | Tester → ECU (upload) | |
| --- | --- | --- | --- |
| Message Type: | | Request | |
| Data byte | Description (all values are in hexadecimal) | | Byte Value (Hex) |
| #1 | TransferData request SID | | 36 |
| #2 | blockSequenceCounter | | 00-FF |

For download, the transferRequestParameterRecord shall include the data to be transferred.

### TransferData positive response

Table 3.69 - TransferData positive response message flow (download)

| Message direction: | | ECU → Tester (download) | |
| --- | --- | --- | --- |
| Message Type: | | Response | |
| Data byte | Description (all values are in hexadecimal) | | Byte Value (Hex) |
| #1 | TransferData response SID | | 76 |
| #2 | blockSequenceCounter | | 00-FF |

Table 3.70 - TransferData positive response message flow (upload)

|  |  |  |  |
| --- | --- | --- | --- |
| Message direction: | | ECU → Tester (upload) | |
| Message Type: | | Response | |
| Data byte | Description (all values are in hexadecimal) | | Byte Value (Hex) |
| #1 | TransferData response SID | | 76 |
| #2 | blockSequenceCounter | | 00-FF |
| #3 | transferResponseParameterRecord[transferResponseParameter#1] =  data#1 | | 00-FF |
| #4 | transferResponseParameterRecord[transferResponseParameter#2] =  data#2 | | 00-FF |
| #5 | transferResponseParameterRecord[transferResponseParameter#3] =  data#3 | | 00-FF |
| : | : | | : |
| #n | transferResponseParameterRecord[transferResponsetParameter#n] =  data#n | | 00-FF |

For upload, the transferResponseParameterRecord shall include the data to be transferred.

### Supported Negative Response Codes

See [ISO 14229-1] for TransferData supported negative response codes.

### TransferData "Early Acknowledge" Strategy (Pipelining)

In order to optimize the bus utilization (especially when downloading to a single ECU), ECUs shall implement an "Early Acknowledge" strategy for service 36H (TransferData). With previous SWDL specifications, when an expected TransferData request was received, the ECU would write the data to memory, and then send a positive response. When using that approach, while the ECU is writing data to memory the bus is essentially quiet (assuming parallel programming of multiple modules is not occurring). The "Early Acknowledgement" strategy is intended to better utilize the bus bandwidth during these scenarios. Using "Early Acknowledgment" strategy, 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 Core Network Communications.

A simplified example illustrating the difference between the traditional SWDL (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 (36H) request. Program Time is the time required for the ECU to write the data from each transferData (36H) request to memory. For the example, the Transmit Time and Program Time are both assumed to be 100ms.

The traditional SWDL 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" SWDL 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}).

## RequestTransferExit (37H) service

This service is used by the tester to terminate a data transfer between tester and ECU (upload or download).

Table 3.71 - RequestTransferExit request message flow

|  |  |  |  |
| --- | --- | --- | --- |
| Message direction: | |  | |
| Message Type: | |  | |
| Data byte | Description (all values are in hexadecimal) | | Byte Value (Hex) |
| #1 | RequestTransferExit request SID | | 37 |

### RequestTransferExit positive response

Table 3.72 - RequestTransferExit positive response message flow

| Message direction: | | ECU → Tester | |
| --- | --- | --- | --- |
| Message Type: | | Response | |
| Data byte | Description (all values are in hexadecimal) | | Byte Value (Hex) |
| #1 | RequestTransferExit response SID | | 77 |
| #2 | transferResponseParameterRecord[transferResponseParameter#1 ] =  checksum byte 1 (MSB) | | 00-FF |
| #3 | transferResponseParameterRecord[transferResponseParameter#2 ] =  checksum byte 2 (LSB) | | 00-FF |

The transferResponseParameterRecord shall contain a two byte checksum (CRC16-CITT) which is required by the tester to support the transfer of data. The two-byte checksum shall be calculated (see Annex B) including all data bytes specified in the latest RequestDownload (or RequestUpload) service and shall be calculated over the transferred data after the data bytes have been programmed into memory. For transfers using a non-zero dataFormatIdentifier, this requires that any decompression or decryption of the data bytes in each transferData request has been performed prior to calculating the returned checksum and the returned checksum is calculated using these stored data bytes.

If an ECU implements a running checksum of the data received over the network in RAM and a comparison of this RAM checksum to the calculated checksum which occurred over the data after being programmed into memory indicates a programming error, the ECU shall respond with a negative response code of 72H (generalProgrammingError).

If a positive response is returned, it is the responsibility of the tester to compare the ECU's returned checksum in the transferResponseParameterRecord (i.e., the checksum calculated over the data programmed in memory) to the expected checksum and take appropriate action if these do not match (e.g., abort the download sequence).

### Supported Negative Response Codes

See [ISO 14229-1] for RequestTransferExit supported negative response codes.

### Checksum algorithm

The checksum algorithm to be used for the transferResponseParameterRecord in the requestTransferExit positive response (see section 3.10.1) shall be the CRC16-CITT:

* Polynomial: x^16+x^12+x^5+1 (1021H)
* Initial value: FFFFH

For a fast CRC16-CITT calculation a look-up table implementation is the preferred solution. For ECUs with a limited amount of flash memory (or RAM), other implementations may be necessary.

See Annex B for "C-code" example.

## TesterPresent (3EH) service

This service is used to indicate to an ECU (or ECUs) that a tester is still connected to the vehicle and that certain diagnostic services and/or communication that have been previously activated are to remain active.

Table 3.73 - TesterPresent request message flow

|  |  |  |  |
| --- | --- | --- | --- |
| Message direction: | | Tester → ECU | |
| Message Type: | | Request | |
| Data byte | Description (all values are in hexadecimal) | | Byte Value (Hex) |
| #1 | TesterPresent request SID | | 3E |
| #2 | zeroSubFunction | | 00 or 80 |

### TesterPresent positive response

Table 3.74 - TesterPresent positive response message flow

| Message direction: | | ECU → Tester | |
| --- | --- | --- | --- |
| Message Type: | | Response | |
| Data byte | Description (all values are in hexadecimal) | | Byte Value (Hex) |
| #1 | TesterPresent response SID | | 7E |
| #2 | zeroSubFunction | | 00 |

### Supported Negative Response Codes

See [ISO 14229-1] for TesterPresent supported negative response codes.

## Service not supported

For not supported services the following negative response message shall be used. The negative response may be used even if the service is supported outside of the programming session (i.e., in the application software).

Table 3.75 - Service not supported negative response message flow

|  |  |  |  |
| --- | --- | --- | --- |
| Message direction: | | ECU → Tester | |
| Message Type: | | Response | |
| Data byte | Description (all values are in hexadecimal) | | Byte Value (Hex) |
| #1 | Negative Response SID | | 7F |
| #2 | Not supported SID | | xx |
| #3 | responseCode = serviceNotSupported | | 11 |

If the service is known to be supported outside of the programming session, the ECU may alternatively utilize negative response code 7FH (serviceNotSupportedInActiveSession).

# File Download Sequence

The file download method provides a means for an off-board tester to send multiple bytes of data to an ECU. The file download process combines a number of bootloader services to achieve this.

The file download sequence is based on the non-volatile server memory programming process described in   
[ISO 14229-1]. Only the programming step (STP2) and post-programming step (STP3) of Programming Phase #1 is used.

Message sequencing consists of a means for initiating the download, transfer of messages and exit of the download operation as detailed in Figure 4.1 to Figure 4.7 for an ordered list of files.

It is recommended that all released software files which are downloadable to the ECU shall be capable of being downloaded to the ECU individually or in any grouping combination independent of order. The exception is that the secondary bootloader shall always be required to be downloaded first when supported. While it is recommended to keep these to a minimum, any software dependencies (whether within a single ECU or across ECUs) or file download order dependencies shall be captured within the Ford software vault using the processes required by Ford's IVS department. All dependencies for any software part shall be agreed to and verified to be correctly documented within IVS between the supplier and the Ford design and release engineer.

Examples of potential software dependencies based upon functional linkage or software architecture decisions are:

1. Dependency Across ECUs:

When the Engine Control Module software is updated to suffix 'BC', the TCM software must be updated to suffix 'DG'.

1. Dependency Within One ECU:

When the strategy is updated within an ECU, the calibration must always be reprogrammed as well.

1. File Order Dependency:

When two calibration parts are downloaded to an ECU, calibration part #1 must be downloaded prior to downloading calibration part #2.

## Active Partition Download Programming Sequence - Recommended

The sequence described in this section is strongly recommended to ensure the software download procedure implemented in the tester is compatible with all ECUs. To ensure optimal programming times and prevent false DTCs, functional requests shall be sent on all public networks at the vehicle data link connector. Note that the periodic transmittal of the functional DiagnosticSessionControl is intended to ensure all ECUs correctly enter programmingSession, especially on vehicle architectures where ECUs running their application may selectively disable power to other ECUs despite a vehicle ignition status equivalent to RUN. In this situation, the disabled ECUs will not receive the DiagnosticSessionControl requests until they are powered (e.g., through the controlling ECU's PBL). Note that pre-programming step (STP1) of Programming Phase #1 in [ISO 14229-1] is not utilized.



Figure 4.1 Recommended service/end-of-line programming sequence (1 of 3)



Figure 4.2 Recommended service/end-of-line programming sequence (2 of 3)



Figure 4.3 Recommended service/end-of-line programming sequence (3 of 3)

## Active Partition Download Programming Sequence - Minimal

The minimum programming sequence contains the minimum steps a tester can take and still guarantee successfully programming of any ECU compliant to this specification.



Figure 4.4 Minimum programming sequence (1 of 2)



Figure 4.5 Minimum programming sequence (2 of 2)

## Download Data Block

The maximum amount of data that can be specified by a single RequestDownload (SID 34H) is 4 gigabytes.



Figure 4.6 Download Data Block (1 of 2)



Figure 4.7 Download Data Block (2 of 2)

## Inactive Partition Download Programming Sequence

The table below compares the high level sequence for the normal active partition programming and compares that with the typical sequence that would be used when programming the inactive partition (e.g., for OTA purposes), as well as the sequence for rolling back to the previous active software.

|  |  |  |  |
| --- | --- | --- | --- |
| Step | Active Partition Programming | Inactive Partition Programming | Inactive Partition Rollback |
| 1 | Enter programming session | Enter programming session | Enter programming session |
| 2 | Unlock ECU | Unlock ECU | Unlock ECU |
| 3 | Download SBL | Download SBL | Download SBL |
| 4 |  | Set OTA Update State (021BH) =  Inactive partition update state | Set OTA Update State (021BH) =  Rollback state  See Note 2 |
| 5 |  |  | Calculate SWash (0219 H)  See Note 1 |
| 6 | Activate SBL (0301H) | Activate SBL (0301H) | Activate SBL (0301H) |
| 7 | Erase 1st file memory | Erase 1st file memory |  |
| 8 | Program 1st file memory | Program 1st file memory |  |
| 9 | : | : |  |
| 10 | Erase Nth file memory | Erase Nth file memory |  |
| 11 | Program Nth file memory | Program Nth file memory |  |
| 12 |  | Prepare Inactive Memory (021AH) |  |
| 13 | Validate application (0304H) | Validate application (0304H) |  |
| 14 |  | Calculate Swash (0219H)  See Note 1 |  |
| 15 |  | Replace Active Memory Partition (021CH) | Replace Active Memory Partition (021CH) |
| 16 | ECU Reset | ECU Reset | ECU Reset |

Note 1. Sending of the SWash routine is optional and not a required step. The benefit is that it allows the tester to confirm exactly what software will be activated if routine 021C is successful.

Note 2. This step can be performed before or after download of the SBL. Sending this check and the SWash calculation before downloading the SBL would prevent unnecessary downloading of the SBL in case the SWash value does not match expectations.

Except where otherwise noted (e.g., Prepare Inactive Memory 012AH can’t be performed without first activating the SBL while the current OTA update state equal “Inactive partition update state”), the steps in the inactive partition programming and rollback do not have to be performed during the same instantiation of the programmingSession. As an example, assume ECU A and ECU B are undergoing a coordinated update and the inactive partition is being programmed. A valid sequence of events is as follows:

Vehicle is inhibited and all ECUs placed into programmingSession

ECU A is unlocked, SBL is downloaded, and memory for file 1 is erased and programmed into inactive memory.

ECU A begins erasing and download for file 2, but customer cancels the OTA event and vehicle is de-inhibited

The next day vehicle OTA campaign resumes. Vehicle is inhibited and ECUs placed into programmingSession.

ECU A is unlocked, SBL is downloaded, and memory for file 2 is erased and programmed into inactive memory. File 1 was already successfully programmed.

ECU A successfully performs prepare inactive memory and validate application over the inactive memory partition.

Battery state of charge limitation causes the current OTA update sequence to abort

The next day vehicle OTA campaign resumes. Vehicle is inhibited and ECUs placed into programmingSession.

ECU B is unlocked, SBL is downloaded, and memory for file 1 is erased and programmed into inactive memory.

ECU B successfully performs prepare inactive memory and validate application over the inactive memory partition.

ECU A is unlocked and SBL is downloaded and activated with OTA update state equal to “Inactive partition update state”

OTA tester performs a SWash check on ECU A and ECU B to validate the contents of their inactive partitions

ECU A and ECU B perform Replace Active Memory Partition and are reset by the OTA tester

OTA tester reads part numbers and discovers OTA B is still running old software

Due to coordinated software, ECU A must be rolled back, so all ECUs placed into programmingSession

ECU A is unlocked and SBL is downloaded and activated with OTA update state equal to “Rollback state”

OTA tester performs a SWash check on ECU A to ensure contents of the active backup to be used for rollback.

ECU A performs Replace Active Memory Partition and all ECUs are reset by the OTA tester

OTA tester reads part numbers on ECU A to verify it rolled back to original software

OTA update flagged as unsuccessful (no change) and vehicle de-inhibited.

# File Upload Sequence

The file upload method provides a means for an ECU to send multiple bytes of data to an off-board tester. The file upload process combines a number of bootloader services to achieve this.

Message sequencing consists of a means for initiating the upload, transfer of messages and exit of the upload operation as detailed in Figure 5.1 to Figure 5.5.

## Upload Programming Sequence



Figure 5.1 File upload procedure (1 of 5)



Figure 5.2 File upload procedure (2 of 5)



Figure 5.3 File upload procedure (3 of 5)

## Upload Data Block

The maximum amount of data that can be specified by a single RequestUpload (SID 35H) is 4 gigabytes.



Figure 5.4 File upload procedure (4 of 5)



Figure 5.5 File upload procedure (5 of 5)

# ECU Program Mode

Whenever an ECU is executing the programmingSession or bootloader defaultSession, it shall ensure the transmission of all normal inter-module communication and network management messages is suspended. Additionally, the ECU shall ensure the monitoring and logging of all diagnostic trouble codes (DTCs) is disabled. The ECU shall resume all normal inter-module communication, network management communication, and DTC logging as determined by the ECU control strategy when the ECU exits the programmingSession and begins executing a different diagnostic session within the ECU application.

When a valid DiagnosticSessionControl service with diagnosticSessionType equal to programmingSession is received while the ECU's normal application is executing and any pre-requisite conditions are satisfied, the ECU shall begin executing the PBL programmingSession. The ECU is allowed to perform a reset when transitioning from the ECU normal application to the programmingSession, but shall always be able to receive, interpret and execute requests and send responses to every correct request after a maximum delay of 500 ms following this diagnostic request. When sending a positive response for this request, the ECU shall behave in either of the following ways:

* Send the positive response from the ECU application prior to entering the PBL.
  + If this method is implemented it is possible that the ECU application does not know the P2Server\_max and P2\*Server\_max used by the programmingSession. If the application is unsure of the exact values used by the programmingSession, the application shall utilize values of 25ms and 5000ms respectively. Note that when following the recommended file download sequence as described in section 4, the ECU being programmed will always be in the programmingSession when a request is sent with suppressPosRspMsgIndicationBit = false and will therefore always report the correct values.
* Send the positive response immediately upon entering the PBL.
  + In this case, the ECU will need to ensure it can meet necessary timing requirements (e.g., P2) and may need to store the state of the SPRMIB in order to determine whether a response is required or not upon entering the PBL.

Regardless of which of the above methods is used, the ECU shall be responsible for meeting all timing requirements that apply as specified in this specification and [Ford GGDS].

## ECU Program Mode for Bootloader Implementations

The state diagram below shows examples of the different states an ECU can be in. In all these states it shall be possible to force the ECU into program mode with the DiagnosticSessionControl(programmingSession) service.

Program mode means that the ECU exits the normal application and starts executing the primary bootloader.



Figure 6.1 Program mode

If the ECU does not have any application software, the ECU shall stay in program mode as shown in Figure 6.1. It is possible that an ECU can get into an unknown state in which the ECU does not have the possibility to determine whether it is programmed or not (e.g., the ECU believes it has a valid application programmed when it does not). The only way to re-program this ECU is to power it up while the tester sends out the DiagnosticSessionControl (programmingSession) service (see section 6.1.1).

The PBL can be in two different states or sessions, the defaultSession or the programmingSession. After power on the PBL shall begin an initialization phase where it waits for a DiagnosticSesssionControl service. The defaultSession within the PBL can only be entered when no valid application software is present, and a timeout occurs from the initialization phase. Note that the defaultSession within the bootloader is **not** the same as the defaultSession within the application. When the PBL receives the service DiagnosticSessionControl with diagnosticSessionType equal to programmingSession it shall switch to programmingSession.

### Boot Sequence for ECUs

The primary bootloader (PBL) shall have a time window of 20 ms (±10%) for detection of the DiagnosticSessionControl service. If a DiagnosticSessionControl service with diagnosticSessionType equal to programmingSession is received during the 20 ms time window after power on (either physically or functionally addressed and either with SPRMIB = true or SPRMIB = false), the ECU shall enter programmingSession. If not, the ECU shall exit the PBL and jump to the application (assuming a valid application has been deemed present).

The ECU shall ignore any request other than the DiagnosticSessionControl service with diagnosticSessionType equal to programmingSession during the 20 ms time window after power on.

This backdoor solution shall be used if the ECU believes it has a valid application software present, but the application software is not correctly jumping to the PBL.

The tester shall send out the DiagnosticSessionControl service with diagnosticSessionType set to programmingSession and suppressPosRspMsgIndicationBit equal to TRUE during a minimum time of 2 seconds. The repetition rate of the DiagnosticSessionControl service sent from the tester shall be 5 ms or faster.

Figure 6.2 shows a typical boot sequence of ECUs in a vehicle.



Figure 6.2 Boot sequence

## Validation of ECU Software

The bootloader shall be responsible for determining whether or not a valid software application is present or not present prior to executing new software. To achieve this, the bootloader shall be responsible for the logical implementation of the following three software functions. Note that the functions below do not mandate any specific implementation (e.g., application signing) or application programming interface, but rather provide direction in order to help logically determine whether a valid application is present. The actual mechanism for validating that the downloaded software is valid and authentic for the particular hardware shall be determined based upon agreement between the diagnostic feature owner and the implementer.

1. Is Valid Application Present

This function shall be called whenever the primary bootloader begins execution after an ECU power up. The function shall return the value of "false" to inform the bootloader that valid application software is not present, thereby requiring the bootloader to begin execution of the PBL defaultSession after the required timeout (see exiting "Init" phase after timeout in Figure 6.1). The function shall return the value "true" to inform the bootloader that valid application software is present, thereby requiring the bootloader to begin execution of the ECU's application after the required timeout from the "Init" phase.

1. Determine Valid Application Status

This function shall be called each time the bootloader receives a validly formatted request to start routineIdentifier 0304H (Check Valid Application). This function may depend upon the presence of a secondary bootloader to execute the needed functionality if necessary. If valid application software is detected (i.e., the bootloader shall pass ECU control over to the application upon the next ECU power up), this function shall set any necessary parameters so that the "Determine Valid Application Status" function will return a value of "true". The details of what makes an application "valid" is specific to each ECU and may be implicitly performed (e.g., embedding a flag at the end of a download) or explicitly performed (e.g., using routineIdentifier 0304H to trigger a validation of a table of necessary modules or even a validation function embedded within the application). At minimum, a valid application should provide the capability to jump back to the bootloader using service 10H (DiagnosticSessionControl).

1. Invalidate Application

If validation of the application is explicitly performed, this function shall ideally be called when a request is made to erase flash memory or program memory that could affect the validity of the application. For example, if the ECU supports downloading to flash memory, this function shall ideally be called each time a request to start controlRoutine FF00H, that could affect the validity of the application, is accepted by the ECU. If the ECU supports downloading to all other non-volatile memory types, this function shall ideally be called each time a requestDownload service is accepted with a memoryAddress within this non-volatile memory range. This may depend upon the presence of a secondary bootloader to execute the needed functionality if necessary. This function shall set any necessary parameters so that the "Determine Valid Application Status" function will return a value of "false", thereby forcing a valid download followed by a routineIdentifier 0304H request to again validate the application.

**NOTE**: A valid application as determined by the bootloader does not necessarily guarantee that the application detected is the correct one or that no additional executable code or calibrations will need to be downloaded in order for the ECU to become fully operational. A valid application as determined by the bootloader only guarantees that the bootloader will begin executing the ECU's normal application after a timeout from the init phase (see Figure 6-1), and will not be "stuck" in the primary bootloader defaultSession.

## Keep ECU programming session

When starting the ECU programming session, the ECU shall start a timer S3Server. The implementation of the S3Server timer (e.g., start and stop requirements) shall be compliant to the requirements in [ISO 14229-2]. The timeout value for S3Server is 5000 ms.

If the maximum value for S3Server is reached, the ECU shall make a hard reset and exit the ECU programming session.

## ECU Connected to Multiple Public Diagnostic Networks

An ECU connected to multiple public networks shall only be downloadable on one of the networks (the primary network). If the ECU supports diagnostic communication on multiple public networks, then a session control for programming session sent either physically or functionally will cause a switch to the PBL. From that point onwards the ECU shall only communicate on the primary network and shall ignore any tester communication on the other networks until the ECU leaves the programming session. Note that this may mean that when a programming sequence is carried out on a single network other than the primary network, the ECU will time out from the programming session and reset after the expiration of the session timer.

Where an ECU connected to multiple public networks supports diagnostic communication on a single network, then it shall support software download on that same network.

# Application Layer

## Timing Parameters

Table 7.1 specifies allowed application layer timing ranges for diagnostic services.

Table 7.1 - Diagnostic service timing parameters

|  |  |  |  |
| --- | --- | --- | --- |
| Parameter | Value | | Description |
|  | Min. | Max. |  |
| P2Server | 0 | 25 ms | Time between tester request message and the receipt of an unsegmented response message or first frame of a segmented response message. The ECU shall start sending its response message within P2Server after the request message has been correctly received. |
| P2\*Server | 0 | 5000 ms | Time between the successful reception of a negative response message with response code 78H and the next response message (positive or negative message). Values of greater than 5000ms are allowed, but only when approved by Ford Core Network Communications. |

The tester shall dynamically adapt to application layer timing parameters for each ECU by sending the DiagnosticSessionControl service with sub-parameter suppressPosRspMsgIndicationBit = FALSE.

The tester may send a new request message immediately after it has determined that the response from the previously sent request message has been received. The target is to use the minimum value for P2.

**NOTE**: A detailed description of P2 and P2\* can be found in [ISO 14229-1] and [ISO 14229-2]. The description of P2 above has been adopted to the fact that only one ECU respond on a request from the tester during a software download operation.

The value used for ΔP2Req shall be zero for all main nodes. The value used for ΔP2Resp shall be 25ms for all main nodes. The values used for ΔP2Req and ΔP2Resp for sub-nodes shall be specified by the implementer but shall never exceed 150ms for either parameter. Note that ΔP2 = ΔP2Req + ΔP2Resp.

## Diagnostic Services Response Times

An ECU shall always execute and complete a request before transmitting a final response unless specified otherwise in this document or in [ISO 14229-1]. If the time required to complete a request is longer than the specified P2Server\_Max then the ECU may respond with negative response code 78H (requestCorrectlyReceived-ResponsePending) as a method of delaying its final response. The ECU shall support the diagnostic service response time P4 as detailed in Table 7.2. P4 is defined in note 2 of Table 7.2. Negative response code 78H is defined in [ISO 14229-1]. An ECU shall execute and complete diagnostic requests within P4.

Note that when negative response code 78H is utilized for a given request, the ECU shall always send its final response to that request (i.e., positive response or negative response message with negative response code other than 78H), independent from the value of the suppressPosRspMsgIndicationBit.

Table 7.2 - Requirements on maximum value of P4.

| Service/Sub-function,  see Note 1 | | P4 max,  see Note 2 |  |
| --- | --- | --- | --- |
| Name | Hex |  | Notes |
| DiagnosticSessionControl | 10 | P2Server\_Max |  |
| ECUReset | 11 | P2Server\_Max |  |
| ReadDataByIdentifier | 22 | P2Server\_Max |  |
| SecurityAccess | 27 | P2Server\_Max |  |
| WriteDataByIdentifier | 2E | 5000 ms |  |
| RoutineControl | 31 | 60000 ms | 3,4,5,6 |
| RequestDownload | 34 | 1000 ms |  |
| RequestUpload | 35 | 1000 ms |  |
| TransferData | 36 | 5000 ms |  |
| RequestTransferExit | 37 | 15000 ms | 5 |
| TesterPresent | 3E | P2Server\_Max |  |

**Notes in Table 7.2:**

1. Note that these requirements are applicable only to services that are supported by the ECU within the programmingSession (refer to [Ford GGDS] for all other services). Exceptions to P4max are allowed but only when approved by Ford Core Network Communications. P4 values for each supported service shall be documented in the ECU's Subsystem Specific Diagnostic Specification.

2. The timing parameter P4 is the time between the reception of a request (N\_USData\_indication) and the start of transmission of the final response (N\_USData\_request). A final response is a positive response or a negative response other than response code 78H "requestCorrectlyReceived-ResponsePending".

P4max is the maximum value of P4. See [ISO 15765-2] for definition of the network layer service primitives and network layer timing parameters.

If P4max is the same as P2Server\_max in the table, this means that a negative response with response code 78H is not allowed.

3. routineIdentifier FF00H (eraseMemory) may increase P4max time 60s for each megabyte range of data over 1 megabyte that is being erased in the request. For example, an eraseMemory request with a memorySize value between 1 and 2 megabytes may utilize a P4max time of 120s and an eraseMemory request with a memorySize value between 2 and 3 megabytes may utilize a P4max time of 180s.

4. routineIdentifier 0304H (Check Valid Application) may utilize a P4max time up to 180s where explicit approval is granted by Ford Core Network Communications and specific agreements are made with all affected tools from EOL and/or service.

5. To account for checksum calculation times used in Service 37H (RequestTransferExit) and Service 31H (RoutineControl) with routineIdentifier FF01H (checkProgrammingDependencies), a given download block may increase P4max time 5s for each 250K of data over 250K that is sent in a single block. For example, a service 37H request used to terminate a single download block under 250K would allow 15s, a service 37H request relating to a data block between 250K and 500K would allow 20s, and a service 37H relating to a data block between 500K and 750k would allow 25s, etc.

6. routineIdentifier 021AH (Prepare Inactive Memory) and routineIdentifier 021CH (Replace Active Memory Partition) may utilize a P4max time up to 120s.

## Requirements for Processing Additional Tester Requests

If the ECU diagnostic application layer receives one or more additional tester requests (physically or functionally addressed) from the network layer before the ECU has completed processing a pending tester request, it shall behave in either of the following ways. Note that the processing of a pending request shall always be regarded as completed at least before the final response is transmitted (independent of whether the requested functionality has completed or not) and within P3Client\_Phys\_min or P3Client\_Func\_min of the pending request if no response is required:

1. Ignore these additional tester requests. However, if the pending request is a functionally addressed testerPresent (3EH) with suppressPosRspMsgIndicationBit equal to true, the ECU shall disregard the pending request and process the first additional tester request.
2. Process as many additional requests as the ECU diagnostic application is able to buffer and still meet all timing requirements (e.g., P2Server\_Max, P4Max, etc.). The requests shall be responded to in the order they were received and a minimum of one additional request shall be capable of being buffered if this approach is used.

# Network Layer

## CAN and CAN FD

Where the ECU supports Diagnostics over CAN or CAN FD, the ECU’s network layer shall comply to standards in accordance with [ISO 15765-2].

### Addressing Format

An ECU shall support both physically and functionally addressed requests from a tester.

#### CAN 11-bit Identifiers

Where the ECU supports Diagnostics over CAN or CAN FD, the ECU shall support 11-bit CAN identifiers using the normal addressing format defined in [ISO 15765-2].

CAN identifiers reserved for addressing ECUs are described in Table 8.1.

An ECU shall support both physically and functionally addressed requests from the tester. CLASSICAL CAN ECUs shall support a total of three (3) USDT CAN diagnostic IDs. One ID is reserved for physically addressed requests and is designated as the ECU Diagnostic Reception ID. One ID is reserved for physically addressed responses and is designated as the ECU Diagnostic Transmission ID. The ECU Transmission ID is always equal to the value of the ECU Reception ID plus eight (8). The third ID is reserved for functionally addressed requests and shall always have the value of 7DFH. The ECU shall always send its responses back to the tester with physical addressing, regardless of whether the request was physically or functionally addressed.

Table 8.1 specifies the 11-bit CAN identifiers to be used.

CAN FD ECUs shall support the CLASSICAL CAN USDT IDs as specified in the preceding paragraph. They shall also support three additional 11-bit CAN FD USDT CAN IDs to be used for the same purposes: one for physically addressed requests, another for physically addressed responses (Transmission ID), and the third for functionally addressed requests, for a total of six (6) CAN IDs. The CAN FD IDs shall have the same values as their CLASSICAL CAN ID counterparts.

Table 8.1 - 11-bit CAN identifiers

|  |  |
| --- | --- |
| CAN identifier (Hex) | Description |
| 640 | Physical request CAN identifier from tester to ECU A |
| 648 | Physical response CAN identifier from ECU A to tester |
| 641 | Physical request CAN identifier from tester to ECU B |
| 649 | Physical response CAN identifier from ECU B to tester |
| : | : |
| 677 | Physical request CAN identifier from tester to ECU C |
| 67F | Physical response CAN identifier from ECU C to tester |
| 6C0 | Physical request CAN identifier from tester to ECU D |
| 6C8 | Physical response CAN identifier from ECU D to tester |
| 6C1 | Physical request CAN identifier from tester to ECU E |
| 6C9 | Physical response CAN identifier from ECU E to tester |
| : | : |
| 7D6 | Physical request CAN identifier from tester to ECU F |
| 7DE | Physical response CAN identifier from ECU F to tester |
| 7DF | CAN identifier to functionally addressed request messages sent by the tester. Used for enter and exit the program mode of the ECUs |
| 7E0 | Physical request CAN identifier from tester to ECU G |
| 7E8 | Physical response CAN identifier from ECU G to tester |
| 7E1 | Physical request CAN identifier from tester to ECU H |
| 7E9 | Physical response CAN identifier from ECU H to tester |
| : | : |
| 7E7 | Physical request CAN identifier from tester to ECU I |
| 7EF | Physical response CAN identifier from ECU I to tester |

### Flow Control

Where the ECU supports Diagnostics over CAN or CAN FD, the following requirements specify network layer parameters and flow control for main nodes as defined in [ISO 15765-2].

#### FlowStatus (FS) parameter

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

### Maximum Number of FC.Wait Frame Transmission (N\_WFTmax)

The value of the N\_WFTmax parameter defined in [ISO 15765-2] for a tester shall be a maximum of 600. The value of the N\_WFTmax parameter for an ECU shall be zero, signifying that the ECU shall never transmit flow control frames with a FlowStatus of Wait (1).

#### BlockSize (BS) Parameter

Flow control frames transmitted from a tester shall have a BlockSize of 00H.

ECUs shall not send additional flow control frames by reporting a BlockSize of 00H in their first flow control frame.

With explicit approval from Ford Network Communications, if an ECU has a separation time STmin greater than 00H and is behind a gateway that cannot preserve the reported STmin, then the ECU may mitigate this issue by specifying a BlockSize of 01H to force a flow control frame between each consecutive frame.

#### SeparationTime (STmin) parameter

Testers shall be able to receive consecutive frames with a separation time of 0 msec and shall report an STmin of 00H in their flow control frame.

ECUs shall minimize the separation time needed to receive consecutive frames and shall report this STmin value in their flow control frame. For ECUs with a reprogrammable memory size greater than 1 megabyte, STmin values greater than 0 must be approved by Ford Core Network Communications.

ECUs shall be able to receive consecutive frames with a separation time of 1 msec or less.

### Timing Parameters

The network layer timing parameter values, timeout and performance requirement values, shall be as detailed in Table 8.2 below. These values shall be used by the tester and the ECU for request frames (functional as well as physical) and response frames. The values are applicable for any ECU implementing software download compliant with this specification.

Table 8.2 - Network layer timeout and performance requirement values

|  |  |  |
| --- | --- | --- |
| Parameter | Timeout value | Performance requirement value |
| N\_As/N\_Ar | 1000 ms | - |
| N\_Bs | 1000 ms |  |
| N\_Br | - | (N\_Br + N\_Ar) < 2250 us @ 500 kbit/s  (N\_Br + N\_Ar) < 3000 us @ 125 kbit/s |
| N\_Cs | - | (N\_Cs + N\_As) < 1250 us @ 500 kbit/s  (N\_Cs + N\_As) < 2000 us @ 125 kbit/s |
| N\_Cr | 1000 ms | - |

**NOTE**: A detailed description of the network layer timing parameter values can be found in [ISO 15765-2].

For high performance during download and upload, the timing parameter values for N\_Br and N\_Cs, shall be as low as possible.

### Buffer Size

The maximum message size shall be limited to 4095 bytes.

Where the ECU supports Diagnostics over CAN or CAN FD, the FirstFrame data length escape sequence of all zeros shall not be supported. If an ECU receives a FirstFrame indicating a size greater than 4095 then it shall send a FlowControl frame with FlowStatus = OVFLW (Overflow) and abort the reception in accordance with [ISO 15765-2].

### TX\_DL

Where the ECU supports Diagnostics over CAN or CAN FD, TX\_DL and its usage is specified in [ISO 15765-2] and defines the maximum frame size that will be used for a message. CAN FD TX\_DL values may be any value allowed by [ISO 15765-2], however the recommended value is 64 bytes. An ECU shall be able to receive messages of any TX\_DL value.

## Ethernet

Where the ECU supports Diagnostics over Internet Protocols, the ECU’s network layer shall comply with all applicable requirements for Internal DoIP Server in accordance with [Ford DoIP].

# Data Link Layer

## CLASSICAL CAN Data Length Code (DLC)

Where the ECU supports Diagnostics over CAN, the CLASSICAL CAN DLC (Data Length Code) contained in every diagnostic CAN frame shall always be set to eight (8). Any diagnostic frame with a DLC other than eight shall be considered invalidly formatted and ignored by the recipient (i.e., no response shall be elicited and no action shall be taken). Any unused data bytes in every diagnostic frame transmitted by the ECU shall be padded with either zeros (00H) or CCH. An ECU shall not reject a received diagnostic frame with other pad bytes values, however.

The data link layer of the tester and ECU shall be compliant with [Ford CAN DL&PHY].

## CAN FD Data Length Code (DLC)

Where the ECU supports Diagnostics over CAN FD, the CAN FD DLC contained in every diagnostic CAN frame shall always be set to a value of eight (8) to fifteen (15). Note that for CAN FD, DLC values of 8, 9, 10, 11, 12, 13, 14, and 15 correspond to frame sizes of 8, 12, 16, 20, 24, 32, 48, and 64 bytes respectively. Any diagnostic frame with a DLC less than eight shall be considered invalidly formatted and ignored by the recipient (i.e., no response shall be elicited, and no action shall be taken).

For frames with DLC = 8 any unused data bytes shall be padded with either zeros (00H) or CCH. For frames with DLC > 8 any unused data bytes shall be padded with CCH to minimize stuff-bit insertions. An ECU shall not reject a received diagnostic frame with other pad byte values, however.

## CLASSICAL CAN and CAN FD

Where the ECU supports Diagnostics over CAN FD, the ECU shall support all diagnostic services on both CLASSICAL CAN and CAN FD. The diagnostic response shall be sent using the same variant of CAN (CLASSICAL CAN or CAN FD) as the corresponding request.

# Explicit Diagnostic Gateways

All software download explicit diagnostic gateways on the main network shall support the requirements detailed in this section for communication to sub-nodes on sub-networks. Explicit diagnostic gateway ECUs have the capability to relay messages from one network to another in order to enable a tester to communicate to ECUs not directly accessible to the tester. The explicit diagnostic gateway transfers the request from the tester to the sub-node and transfers the response from the sub-node back to the tester.

The gateway function shall be implemented in the PBL. Upon entering the programmingSession, the explicit diagnostic gateway shall automatically send the equivalent of a functionally addressed DiagnosticSessionControl (10H), programmingSession with suppressPosRspMsgIndicationBit = TRUE on the sub-network.

**NOTE**: Explicit diagnostic gateways are not the same as transparent diagnostic gateways. Refer to [Ford GGDS] for more information on transparent diagnostic gateways.

## Explicit Gateway State Access

A software download explicit diagnostic gateway is placed into and out of gateway state by utilizing RoutineControl (31H) and routineIdentifer 0300H "Gateway State Access" in the programmingSession. All parameters for routineIdentifier 0300H are specified in section 3.6.1.

RoutineIdentifier 0300H is defined in [Ford GMRDB] as RoutineType 3 and shall support the startRoutine, stopRoutine, and requestRoutineResults sub-functions with RoutineControl (31H) service.

The Request Additional Data for startRoutine shall contain a two-byte address to specify the sub-node that the gateway ECU shall route communication to. The gateway will use this two-byte address to route messages to the appropriate sub-network and physical address. The physical address of an ECU shall be the same regardless of which network or sub-network it is connected to.

* The least significant byte of the two-byte address shall contain the sub-node address. Sub-node addresses are specified in [Ford GMRDB].
* The most significant byte of the two-byte address shall contain the sub-network address. Sub-network addresses are managed in [Ford IVS].

[Ford IVS] tracks combinations of ECU name, acronym, sub-network address, and sub-node address for each program. When an ECU specific sub-node address is used (i.e., E0 H -EF H), the actual program specific ECU name and acronym (not the generic GMRDB name and acronym) shall be used.

## Explicit Gateway State Access Not Active

If the software download explicit diagnostic gateway is not in the gateway state, the following requirements shall apply:

* If the software download explicit diagnostic gateway receives a properly formatted functionally addressed DiagnosticSessionControl (10H), ECUReset (11H) or TesterPresent (3EH) request, the request shall be translated (if necessary) and sent functionally to all sub nodes. All other functionally addressed messages shall not be relayed to the sub-node. After determining the validity of the request to the subnetwork and transmitting the functional request (if applicable) on the sub-network, the software download explicit diagnostic gateway shall process the functional request itself.
* Functionally addressed DiagnosticSessionControl (10H), ECUReset (11H) or TesterPresent (3EH) requests shall have suppressPosRspMsgIndicationBit = TRUE to suppress responses from the sub-nodes. If the tester sends a request with suppressPosRspMsgIndicationBit = FALSE, the software download explicit diagnostic gateway shall change the value of suppressPosRspMsgIndicationBit to TRUE, and send the equivalent of this message on the sub-network.
* The explicit diagnostic gateway shall ignore any responses (positive or negative) sent by sub-nodes caused by functionally addressed DiagnosticSessionControl (10H), ECUReset (11H) or TesterPresent (3EH) requests.

**NOTE**: Exceptions to the requirements for Explicit Gateway State Access Not Active are allowed but only when explicitly approved by Ford Core Network Communications.

## Entering Explicit Gateway State

The software download explicit diagnostic gateway shall enter into gateway state and transmit a positive response when all of the following entry conditions are met:

* The explicit diagnostic gateway is in the programmingSession
* No known sub-node or sub-network fault conditions exist that prevent the software download explicit diagnostic gateway from communicating with the sub-node (e.g., sub-node not alive or open sub-network circuit). If such a fault condition is known to exist the gateway shall send a negative response.
* The software download explicit diagnostic gateway receives validly formatted routineControl request containing a routineIdentifier of 0300H with a sub-function of 01H (startRoutine) and a valid sub-network and sub-node address.

Once the software download explicit diagnostic gateway has entered the gateway state, the following requirements apply:

* All physically addressed request messages received by the software download explicit diagnostic gateway (with the exception of routineControl with routineIdentifier of 0300H) shall be translated (if necessary) and sent to the appropriate sub-node. The software download explicit diagnostic gateway shall also translate (if necessary) response messages received from the sub-node into the tester supported protocol and relay those responses to the tester. Only responses from the sub-node that has been requested to respond shall be transmitted back to the tester and shall be sent on the software download explicit diagnostic gateway's diagnostic transmission ID. The gateway itself shall not act or respond to physically addressed requests on its own behalf while in the gateway state with the exception of the routineIdentifier 0300H requirements detailed in this specification.
* If the software download explicit diagnostic gateway receives a validly formatted routineControl with a sub-function of startRoutine and a routineIdentifier of 0300H, the software download explicit diagnostic gateway shall redirect communication to the new sub-node address as contained within the two byte routineControlOptionRecord and transmit a positive response.
* If the software download explicit diagnostic gateway receives a validly formatted routineControl with a sub-function of stopRoutine and a routineIdentifier of 0300H, the software download explicit diagnostic gateway shall exit the gateway state and continue execution of the programmingSession.
* If the software download explicit diagnostic gateway receives a validly formatted routineControl with a sub-function of requestRoutineResults and a routineIdentifier of 0300H, the software download gateway shall respond with a positive response including a RoutineStatus value of 2 (signifying that the routine is currently active).
* If the S3server (see [ISO 14229-2]) timer expires, the software download explicit diagnostic gateway shall exit the gateway state by performing a hard reset.
* Any and all other exit conditions for the gateway state shall be kept to a minimum, shall be clearly documented in the ECU's Subsystem Specific Diagnostic Specification, and shall require approval by the diagnostic feature owner.
* If the software download explicit diagnostic gateway receives a properly formatted functionally addressed DiagnosticSessionControl (10H), ECUReset (11H) or TesterPresent (3EH) request, the request shall be translated (if necessary) and sent functionally to all sub-nodes. All other functionally addressed messages shall not be relayed to the sub-node.
* Functionally addressed DiagnosticSessionControl (10H), ECUReset (11H) or TesterPresent (3EH) requests shall have suppressPosRspMsgIndicationBit = TRUE to suppress responses from the sub-nodes. If the tester sends a request with suppressPosRspMsgIndicationBit = FALSE, the software download explicit diagnostic gateway shall change the value of suppressPosRspMsgIndicationBit to TRUE, and send the equivalent of this message on the sub-network. After determining the validity of the request to the subnetwork and transmitting the functional request (if applicable) on the sub-network, the software download explicit diagnostic gateway shall process the functional request itself.
* The explicit diagnostic gateway shall ignore any responses (positive or negative) sent by sub-nodes caused by functionally addressed DiagnosticSessionControl (10H), ECUReset (11H) or TesterPresent (3EH) requests.

Note that this specification makes no assumption concerning the protocol or message structure used on the sub-network. It is the responsibility of the explicit diagnostic gateway to perform all necessary translation (e.g., node address, diagnostic message data, baud rate, etc.) to ensure that messages through the gateway are transparent to the tester. The gateway may decide to process individual CAN frames or complete messages. If a frame based explicit gateway is implemented, the size of the first-in first-out queues shall be determined by the diagnostic feature owner based upon an analysis of the use cases of the explicit gateway and implementation details of the sub-network and sub-network ECUs.

## File Download Sequence to Sub-node Using Explicit Gateway

* The tester sends DiagnosticSessionControl request(s) with sub-function of programmingSession and suppressPosRspMsgIndicationBit = TRUE using functional addressing. All ECUs on the main network start to execute their programmingSession.
* The software download explicit diagnostic gateway sends the equivalent of DiagnosticSessionControl with sub-function of programmingSession and suppressPosRspMsgIndicationBit = TRUE using functional addressing automatically to the sub-network. All ECUs on the sub-network start to execute their programmingSession.
* All ECUs on both the main network and the sub-network(s) are now in programmingSession.
* Enter the gateway state with service RoutineControl (31H), routineIdentifier 0300H and startRoutine to address a node on the sub-network.
* Download data to an ECU on the sub-network using the normal download sequence in section 4 File Download Sequence. After entering the explicit diagnostic gateway state, the same file download sequence can be used on both the main network and on the sub-network. However, the tester shall **not** re-send any functionally addressed DiagnosticSessionControl (to prevent the gateway state from be exited), but rather shall start the file download sequence with the physically addressed DiagnosticSessionControl.

# Non-Bootloader ECUs

This section is intended to address additional clarifications, requirements, and/or exceptions for ECUs which do not implement the programmingSession bootloader concept (see section 1.9.1).

## Service 34H Programmable ECU via Application

Refer to section 1.9.1 for a definition of ECUs that meet this criteria. For ECUs that match the definition of ECU Variant #2 or #4, the programmingSession is completely contained within the ECU's normal application strategy. An ECU of this type shall be responsible for meeting all requirements from the sections and subsections specified as applicable below. All sections not explicitly referenced below are not applicable.

* Section 1 “Introduction” provides the purpose / scope, usage of the document, applicable references, and definitions.
* Section 3 “Software Download Services” describes diagnostic services, dataIdentifiers and routines that must be supported and is applicable with the following exceptions
  + P2Server\_max and P2\*Server\_max timing may be the same as for the defaultSession
  + The following dataIdentifiers shall not be supported with service 22H: F180H, F109H
  + The following dataIdentifiers shall be implemented based upon the conditions in Table 2.1: D028H, D03FH
  + The following dataIdentifiers shall not be supported with service 2EH: F111H
  + Routine 0300H shall be optional unless the ECU is required to function as an explicit diagnostic gateway for software download to a sub-network in which case it shall be mandatory.
  + The following routines are optional to be supported: 0301H, 0305H, FF00H, FF01H,
  + The following routines shall not be supported: 021BH, 021CH, 021AH, 0219H,
* Section 4 “File Download Sequence” describes the mechanism for downloading information from the tester to the ECU. This sequence shall be supported with the potential exception of the download and activation of the secondary bootloader.
* Section 5 “File Upload Sequence” describes the mechanism for uploading information from the ECU to the tester. If service 35H is supported, the upload sequence shall be supported with the potential exception of the download and activation of the secondary bootloader.
* Section 6 “ECU Program Mode” describes the transition to and from programming mode and the required functional behaviour within programming mode. The section is applicable with the following exceptions
  + P2Server\_max and P2\*Server\_max timing may be the same as for the default session
  + Subsections 6.1 and 6.2 do not apply
* Sections 7 “Application Layer” and 8 “Network Layer” describe the details required for tester communication and is applicable with the following exceptions
  + The ECU may either utilize the network layer parameters from section 8.1.2 and 8.1.4 or the network layer parameters from [Ford GGDS].
  + The ECU may either utilize the application layer timing parameters from section 7 or the application layer timing parameters from [Ford GGDS].
  + The ECU may additionally support any or all diagnostic services and functionality which is supported in the defaultSession and the extendedDiagnosticSession within this non-bootloader programmingSession
* Section 9 “Data Link Layer” describes requirements relevant to the data link layer.
* Section 10 describes requirements relevant to explicit diagnostic gateways
* Section 11 'Non-Bootloader ECUs' describes requirements relevant to ECUs not implementing the bootloader approach as described in this specification

## Non-Programmable ECU or Non Service 34H Programmable ECU

Refer to section 1.9.1 for a definition of ECUs that meet this criteria. For ECUs that match the definition of ECU Variant #3 or #5, the programmingSession is completely contained within the ECU's normal application strategy. An ECU of this type shall be responsible for meeting all requirements from the sections and subsections specified as applicable below. All sections not explicitly referenced below are not applicable.

* Section 1 “Introduction” provides the purpose / scope, usage of the document, applicable references, and definitions.
* Section 3 “Software Download Services” describes diagnostic services, dataIdentifiers and routines that must be supported and is applicable with the following exceptions
  + P2Server\_max and P2\*Server\_max timing may be the same as for the defaultSession
  + SecurityAccess service 27H with security level 01H (i.e., requestSeed 01H) shall not be supported
  + The following dataIdentifiers shall not be supported with service 22H: F180H, F109H, D028H, D03FH
  + The following dataIdentifiers shall not be supported with service 2EH: F111H
  + Routine 0300H shall be optional unless the ECU is required to function as an explicit diagnostic gateway for software download to a sub-network in which case it shall be mandatory.
  + The following routines shall not be supported: 0301H, 0304H, 0305H, FF00H, FF01H, 021BH, 021CH, 021AH, 0219H,
  + Service 34H (requestDownload), service 36H (transferData) and service 37H (requestTransferExit shall not be supported.
* Section 6 “ECU Program Mode” describes the transition to and from programming mode and the required functional behaviour within programming mode. The section is applicable with the following exceptions
  + P2Server\_max and P2\*Server\_max timing may be the same as for the default session
  + Subsections 6.1 and 6.2 do not apply
* Sections 7 “Application Layer” and 8 “Network Layer” describe the details required for tester communication and is applicable with the following exceptions
  + The ECU may either utilize the network layer parameters from section 8.1.2 and 8.1.4 or the network layer parameters from [Ford GGDS].
  + The ECU may either utilize the application layer timing parameters from section 7 or the application layer timing parameters from [Ford GGDS].
  + The ECU may additionally support any or all diagnostic services and functionality which is supported in the defaultSession and the extendedDiagnosticSession within this non-bootloader programmingSession
* Section 9 “Data Link Layer” describes requirements relevant to the data link layer.
* Section 10 describes requirements relevant to explicit diagnostic gateways
* Section 11 'Non-Bootloader ECUs' describes requirements relevant to ECUs not implementing the bootloader approach as described in this specification

1. : Example Link Traffic
   1. File Download

The example traces for download are based upon the flowcharts from chapter 4 File Download Sequence.

* F = Functional addressing, P = Physical addressing.
* S.F = Single Frame, F.F = First Frame, C.F = Consecutive Frame.
* Zeros in grey are padded, not part of the service.

**Start ECU programming session, identify the ECU and enter security access mode**

| Tester | Description | Data | | | | | | | | Explanation |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Tx (F)** | **DiagnosticSessionControl (S.F)** | **02** | **10** | **82** | **00** | **00** | **00** | **00** | **00** | Start programmingSession |
| **Tx (F)** | **DiagnosticSessionControl (S.F)** | **02** | **10** | **82** | **00** | **00** | **00** | **00** | **00** | Start programmingSession |
| **Tx (F)** | **DiagnosticSessionControl (S.F)** | **02** | **10** | **82** | **00** | **00** | **00** | **00** | **00** | Start programmingSession |
| **Tx (F)** | **DiagnosticSessionControl (S.F)** | **02** | **10** | **82** | **00** | **00** | **00** | **00** | **00** | Start programmingSession |
| **Tx (F)** | **DiagnosticSessionControl (S.F)** | **02** | **10** | **82** | **00** | **00** | **00** | **00** | **00** | Start programmingSession |
| **Tx (F)** | **DiagnosticSessionControl (S.F)** | **02** | **10** | **82** | **00** | **00** | **00** | **00** | **00** | Start programmingSession |
| **Tx (F)** | **DiagnosticSessionControl (S.F)** | **02** | **10** | **82** | **00** | **00** | **00** | **00** | **00** | Start programmingSession |
| **Tx (F)** | **DiagnosticSessionControl (S.F)** | **02** | **10** | **82** | **00** | **00** | **00** | **00** | **00** | Start programmingSession |
| **Tx (F)** | **DiagnosticSessionControl (S.F)** | **02** | **10** | **82** | **00** | **00** | **00** | **00** | **00** | Start programmingSession |
| **Tx (F)** | **DiagnosticSessionControl (S.F)** | **02** | **10** | **82** | **00** | **00** | **00** | **00** | **00** | Start programmingSession |
| **Tx (F)** | **DiagnosticSessionControl (S.F)** | **02** | **10** | **82** | **00** | **00** | **00** | **00** | **00** | Start programmingSession |
| **Tx (F)** | **DiagnosticSessionControl (S.F)** | **02** | **10** | **82** | **00** | **00** | **00** | **00** | **00** | Start programmingSession |
| **Tx (F)** | **DiagnosticSessionControl (S.F)** | **02** | **10** | **82** | **00** | **00** | **00** | **00** | **00** | Start programmingSession |
| **Tx (F)** | **DiagnosticSessionControl (S.F)** | **02** | **10** | **82** | **00** | **00** | **00** | **00** | **00** | Start programmingSession |
| **Tx (F)** | **DiagnosticSessionControl (S.F)** | **02** | **10** | **82** | **00** | **00** | **00** | **00** | **00** | Start programmingSession |
| **Tx (F)** | **DiagnosticSessionControl (S.F)** | **02** | **10** | **82** | **00** | **00** | **00** | **00** | **00** | Start programmingSession |
| **Tx (F)** | **DiagnosticSessionControl (S.F)** | **02** | **10** | **82** | **00** | **00** | **00** | **00** | **00** | Start programmingSession |
| **Tx (F)** | **DiagnosticSessionControl (S.F)** | **02** | **10** | **82** | **00** | **00** | **00** | **00** | **00** | Start programmingSession |
| **Tx (F)** | **DiagnosticSessionControl (S.F)** | **02** | **10** | **82** | **00** | **00** | **00** | **00** | **00** | Start programmingSession |
| **Tx (F)** | **DiagnosticSessionControl (S.F)** | **02** | **10** | **82** | **00** | **00** | **00** | **00** | **00** | Start programmingSession |
|  |  |  |  |  |  |  |  |  |  |  |
| **Tx (P)** | **DiagnosticSessionControl (S.F)** | **02** | **10** | **02** | **00** | **00** | **00** | **00** | **00** |  |
| Rx (P) | Positive response (S.F) | 06 | 50 | 02 | xx | xx | yy | yy | 00 | Application layer timing parameters |
|  |  |  |  |  |  |  |  |  |  |  |
| **Tx (P)** | **ReadDataByIdentifier F111**H **(S.F)** | **03** | **22** | **F1** | **11** | **00** | **00** | **00** | **00** | Request ECUCoreAssemblyNumber |
| Rx (P) | Positive response (F.F) | 10 | 1B | 62 | F1 | 11 | xx | xx | xx | ECUCoreAssemblyNumber, 24 bytes |
| **Tx (P)** | **Flow control** | **30** | **00** | **00** | **00** | **00** | **00** | **00** | **00** |  |
| Rx (P) | Positive response (C.F) | 21 | xx | xx | xx | xx | xx | xx | xx |  |
| Rx (P) | Positive response (C.F) | 22 | xx | xx | xx | xx | xx | xx | xx |  |
| Rx (P) | Positive response (last C.F) | 23 | xx | xx | xx | xx | xx | xx | xx |  |
|  |  |  |  |  |  |  |  |  |  |  |
| **Tx (P)** | **ReadDataByIdentifier F18C**H **(S.F)** | **03** | **22** | **F1** | **8C** | **00** | **00** | **00** | **00** | Request ECU Serial Number |
| Rx (P) | Positive response (F.F) | 10 | 13 | 62 | F1 | 8C | xx | xx | xx | ECU Serial Number, 16 bytes |
| **Tx (P)** | **Flow control** | **30** | **00** | **00** | **00** | **00** | **00** | **00** | **00** |  |
| Rx (P) | Positive response (C.F) | 21 | xx | xx | xx | xx | xx | xx | xx |  |
| Rx (P) | Positive response (last C.F) | 22 | xx | xx | xx | xx | xx | xx | 00 |  |
|  |  |  |  |  |  |  |  |  |  |  |
| **Tx (P)** | **SecurityAccess (S.F)** | **02** | **27** | **01** | **00** | **00** | **00** | **00** | **00** | Request seed |
| Rx (P) | Positive response | 05 | 67 | 01 | xx | xx | xx | 00 | 00 | Seed, (e.g., seed length = 3 bytes) |
| **Tx (P)** | **SecurityAccess (S.F)** | **05** | **27** | **02** | **xx** | **xx** | **xx** | **00** | **00** | Key, (e.g., key length = 3 bytes) |
| Rx (P) | Positive response | 02 | 67 | 02 | 00 | 00 | 00 | 00 | 00 | Security access granted |

**Download the SBL**

| Tester | Description | Data | | | | | | | | Explanation |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Tx (P)** | **RequestDownload (F.F)** | **10** | **0B** | **34** | **00** | **44** | **xx** | **xx** | **xx** | Request download, first data block |
| Rx (P) | Flow control | 30 | 00 | 00 | 00 | 00 | 00 | 00 | 00 |  |
| **Tx (P)** | **RequestDownload (C.F)** | **21** | **xx** | **yy** | **yy** | **yy** | **yy** | **00** | **00** |  |
| Rx (P) | Positive response | 04 | 74 | 20 | 0x | xx | 00 | 00 | 00 | maxNumberOfBlockLength |
|  |  |  |  |  |  |  |  |  |  |  |
| **Tx (P)** | **TransferData (F.F)** | **1x** | **xx** | **36** | **01** | **xx** | **xx** | **xx** | **xx** | blockSequenceCounter = 1 |
| Rx (P) | Flow control | 30 | 00 | 00 | 00 | 00 | 00 | 00 | 00 |  |
| **Tx (P)** | **TransferData(C.F)** | **21** | **xx** | **xx** | **xx** | **xx** | **xx** | **xx** | **xx** |  |
| : | : |  |  |  |  |  |  |  |  |  |
| **Tx (P)** | **TransferData(last C.F)** | **2x** | **xx** | **xx** | **xx** | **xx** | **xx** | **xx** | **xx** |  |
| Rx (P) | Positive response | 02 | 76 | 01 | 00 | 00 | 00 | 00 | 00 | blockSequenceCounter = 1 |
| : | : |  |  |  |  |  |  |  |  |  |
| : | : |  |  |  |  |  |  |  |  |  |
| **Tx (P)** | **TransferData (F.F)** | **1x** | **xx** | **36** | **nn** | **xx** | **xx** | **xx** | **xx** |  |
| Rx (P) | Flow control | 30 | 00 | 00 | 00 | 00 | 00 | 00 | 00 |  |
| **Tx (P)** | **TransferData(C.F)** | **21** | **xx** | **xx** | **xx** | **xx** | **xx** | **xx** | **xx** |  |
| : | : |  |  |  |  |  |  |  |  |  |
| **Tx (P)** | **TransferData(last C.F)** | **2x** | **xx** | **xx** | **xx** | **xx** | **xx** | **xx** | **xx** |  |
| Rx (P) | Positive response | 02 | 76 | nn | 00 | 00 | 00 | 00 | 00 |  |
|  |  |  |  |  |  |  |  |  |  |  |
| **Tx (P)** | **RequestTransferExit (S.F)** | **01** | **37** | **00** | **00** | **00** | **00** | **00** | **00** |  |
| Rx (P) | Negative response | 03 | 7F | 37 | 78 | 00 | 00 | 00 | 00 | Response pending (if necessary) |
| Rx (P) | Positive response | 03 | 77 | xx | xx | 00 | 00 | 00 | 00 | 2-byte CRC |
| : |  |  |  |  |  |  |  |  |  |  |
| **Tx (P)** | **RequestDownload (F.F)** | **10** | **0B** | **34** | **00** | **44** | **xx** | **xx** | **xx** | Request download, next data block |
| Rx (P) | Flow control | 30 | 00 | 00 | 00 | 00 | 00 | 00 | 00 |  |
| **Tx (P)** | **RequestDownload (C.F)** | **21** | **xx** | **yy** | **yy** | **yy** | **yy** | **00** | **00** |  |
| Rx (P) | Positive response | 04 | 74 | 20 | 0x | xx | 00 | 00 | 00 | maxNumberOfBlockLength |
|  |  |  |  |  |  |  |  |  |  |  |
| **Tx (P)** | **TransferData (F.F)** | **1x** | **xx** | **36** | **01** | **xx** | **xx** | **xx** | **xx** | blockSequenceCounter = 1 |
| Rx (P) | Flow control | 30 | 00 | 00 | 00 | 00 | 00 | 00 | 00 |  |
| **Tx (P)** | **TransferData(C.F)** | **21** | **xx** | **xx** | **xx** | **xx** | **xx** | **xx** | **xx** |  |
| : | : |  |  |  |  |  |  |  |  |  |
| **Tx (P)** | **TransferData(last C.F)** | **2x** | **xx** | **xx** | **xx** | **xx** | **xx** | **xx** | **xx** |  |
| Rx (P) | Positive response | 02 | 76 | 01 | 00 | 00 | 00 | 00 | 00 | blockSequenceCounter = 1 |
| : | : |  |  |  |  |  |  |  |  |  |
| : | : |  |  |  |  |  |  |  |  |  |
| **Tx (P)** | **TransferData (F.F)** | **1x** | **xx** | **36** | **nn** | **xx** | **xx** | **xx** | **xx** |  |
| Rx (P) | Flow control | 30 | 00 | 00 | 00 | 00 | 00 | 00 | 00 |  |
| **Tx (P)** | **TransferData(C.F)** | **21** | **xx** | **xx** | **xx** | **xx** | **xx** | **xx** | **xx** |  |
| : | : |  |  |  |  |  |  |  |  |  |
| **Tx (P)** | **TransferData(last C.F)** | **2x** | **xx** | **xx** | **xx** | **xx** | **xx** | **xx** | **xx** |  |
| Rx (P) | Positive response | 01 | 76 | nn | 00 | 00 | 00 | 00 | 00 |  |
|  |  |  |  |  |  |  |  |  |  |  |
| **Tx (P)** | **RequestTransferExit (S.F)** | **01** | **37** | **00** | **00** | **00** | **00** | **00** | **00** |  |
| Rx (P) | Negative response | 03 | 7F | 37 | 78 | 00 | 00 | 00 | 00 | Response pending (if necessary) |
| Rx (P) | Positive response | 03 | 77 | xx | xx | 00 | 00 | 00 | 00 | 2-byte CRC |
| : | : |  |  |  |  |  |  |  |  |  |
| : | : |  |  |  |  |  |  |  |  |  |
| **Tx (P)** | **RoutineControl 0301H (F.F)** | **10** | **08** | **31** | **01** | **03** | **01** | **xx** | **xx** | Activate SBL |
| Rx (P) | Flow control | 30 | 00 | 00 | 00 | 00 | 00 | 00 | 00 |  |
| **Tx (P)** | **RoutineControl 0301H (C.F)** | **21** | **xx** | **xx** | **00** | **00** | **00** | **00** | **00** |  |
| Rx (P) | Positive response | 05 | 71 | 01 | 03 | 01 | 10 | 00 | 00 |  |

**Erase flash**

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Tester | Description | Data | | | | | | | | Explanation |
| **Tx (P)** | **RoutineControl FF00\H (F.F)** | **10** | **0C** | **31** | **01** | **FF** | **00** | **xx** | **xx** | Flash erase, first block |
| Rx (P) | Flow control | 30 | 00 | 00 | 00 | 00 | 00 | 00 | 00 |  |
| **Tx (P)** | **RoutineControl FF00H (C.F)** | **21** | **xx** | **xx** | **yy** | **yy** | **yy** | **yy** | **00** |  |
| Rx (P) | Negative response | 03 | 7F | 31 | 78 | 00 | 00 | 00 | 00 | Response pending (if necessary) |
| : | : |  |  |  |  |  |  |  |  |  |
| Rx (P) | Negative response | 03 | 7F | 31 | 78 | 00 | 00 | 00 | 00 |  |
| Rx (P) | Positive response | 05 | 71 | 01 | FF | 00 | 10 | 00 | 00 |  |
| : | : |  |  |  |  |  |  |  |  |  |
| **Tx (P)** | **RoutineControl FF00H (F.F)** | **10** | **0C** | **31** | **01** | **FF** | **00** | **xx** | **xx** | Flash erase, next block |
| Rx (P) | Flow control | 30 | 00 | 00 | 00 | 00 | 00 | 00 | 00 |  |
| **Tx (P)** | **RoutineControl FF00H (C.F)** | **21** | **xx** | **xx** | **yy** | **yy** | **yy** | **yy** | **00** |  |
| Rx (P) | Negative response | 03 | 7F | 31 | 78 | 00 | 00 | 00 | 00 | Response pending (if necessary) |
| : | : |  |  |  |  |  |  |  |  |  |
| Rx (P) | Negative response | 03 | 7F | 31 | 78 | 00 | 00 | 00 | 00 |  |
| Rx (P) | Positive response | 05 | 71 | 01 | FF | 00 | 10 | 00 | 00 |  |

**Download data**

| Tester | Description | Data | | | | | | | | Explanation |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Tx (P)** | **RequestDownload (F.F)** | **10** | **0B** | **34** | **00** | **44** | **xx** | **xx** | **xx** | Request download, first data block |
| Rx (P) | Flow control | 30 | 00 | 00 | 00 | 00 | 00 | 00 | 00 |  |
| **Tx (P)** | **RequestDownload (C.F)** | **21** | **xx** | **yy** | **yy** | **yy** | **yy** | **00** | **00** |  |
| Rx (P) | Positive response | 04 | 74 | 20 | 0x | xx | 00 | 00 | 00 | maxNumberOfBlockLength |
|  |  |  |  |  |  |  |  |  |  |  |
| **Tx (P)** | **TransferData (F.F)** | **1x** | **xx** | **36** | **01** | **xx** | **xx** | **xx** | **xx** | blockSequenceCounter = 1 |
| Rx (P) | Flow control | 30 | 00 | 00 | 00 | 00 | 00 | 00 | 00 |  |
| **Tx (P)** | **TransferData(C.F)** | **21** | **xx** | **xx** | **xx** | **xx** | **xx** | **xx** | **xx** |  |
| : | : |  |  |  |  |  |  |  |  |  |
| **Tx (P)** | **TransferData(last C.F)** | **2x** | **xx** | **xx** | **xx** | **xx** | **xx** | **xx** | **xx** |  |
| Rx (P) | Negative response | 03 | 7F | 36 | 78 | 00 | 00 | 00 | 00 | Response pending (if necessary) |
| Rx (P) | Positive response | 02 | 76 | 01 | 00 | 00 | 00 | 00 | 00 | blockSequenceCounter = 1 |
| : | : |  |  |  |  |  |  |  |  |  |
| : | : |  |  |  |  |  |  |  |  |  |
| **Tx (P)** | **TransferData (F.F)** | **1x** | **xx** | **36** | **nn** | **xx** | **xx** | **xx** | **xx** |  |
| Rx (P) | Flow control | 30 | 00 | 00 | 00 | 00 | 00 | 00 | 00 |  |
| **Tx (P)** | **TransferData(C.F)** | **21** | **xx** | **xx** | **xx** | **xx** | **xx** | **xx** | **xx** |  |
| : | : |  |  |  |  |  |  |  |  |  |
| **Tx (P)** | **TransferData(last C.F)** | **2x** | **xx** | **xx** | **xx** | **xx** | **xx** | **xx** | **xx** |  |
| Rx (P) | Negative response | 03 | 7F | 36 | 78 | 00 | 00 | 00 | 00 | Response pending (if necessary) |
| Rx (P) | Positive response | 02 | 76 | nn | 00 | 00 | 00 | 00 | 00 |  |
|  |  |  |  |  |  |  |  |  |  |  |
| **Tx (P)** | **RequestTransferExit (S.F)** | **01** | **37** | **00** | **00** | **00** | **00** | **00** | **00** |  |
| Rx (P) | Negative response | 03 | 7F | 37 | 78 | 00 | 00 | 00 | 00 | Response pending (if necessary) |
| Rx (P) | Positive response | 03 | 77 | xx | xx | 00 | 00 | 00 | 00 | 2-byte CRC |
|  |  |  |  |  |  |  |  |  |  |  |
| **Tx (P)** | **RequestDownload (F.F)** | **10** | **0B** | **34** | **00** | **44** | **xx** | **xx** | **xx** | Request download, next data block |
| Rx (P) | Flow control | 30 | 00 | 00 | 00 | 00 | 00 | 00 | 00 |  |
| **Tx (P)** | **RequestDownload (C.F)** | **21** | **xx** | **yy** | **yy** | **yy** | **yy** | **00** | **00** |  |
| Rx (P) | Positive response | 04 | 74 | 20 | 0x | xx | 00 | 00 | 00 | maxNumberOfBlockLength |
|  |  |  |  |  |  |  |  |  |  |  |
| **Tx (P)** | **TransferData (F.F)** | **1x** | **xx** | **36** | **01** | **xx** | **xx** | **xx** | **xx** | blockSequenceCounter = 1 |
| Rx (P) | Flow control | 30 | 00 | 00 | 00 | 00 | 00 | 00 | 00 |  |
| **Tx (P)** | **TransferData(C.F)** | **21** | **xx** | **xx** | **xx** | **xx** | **xx** | **xx** | **xx** |  |
|  | : |  |  |  |  |  |  |  |  |  |
| **Tx (P)** | **TransferData(last C.F)** | **2x** | **xx** | **xx** | **xx** | **xx** | **xx** | **xx** | **xx** |  |
| Rx (P) | Negative response | 03 | 7F | 36 | 78 | 00 | 00 | 00 | 00 | Response pending (if necessary) |
| Rx (P) | Positive response | 02 | 76 | 01 | 00 | 00 | 00 | 00 | 00 |  |
| : | : |  |  |  |  |  |  |  |  |  |
| : | : |  |  |  |  |  |  |  |  |  |
| **Tx (P)** | **TransferData (F.F)** | **1x** | **xx** | **36** | **nn** | **xx** | **xx** | **xx** | **xx** |  |
| Rx (P) | Flow control | 30 | 00 | 00 | 00 | 00 | 00 | 00 | 00 |  |
| **Tx (P)** | **TransferData(C.F)** | **21** | **xx** | **xx** | **xx** | **xx** | **xx** | **xx** | **xx** |  |
| : | : |  |  |  |  |  |  |  |  |  |
| **Tx (P)** | **TransferData(last C.F)** | **2x** | **xx** | **xx** | **xx** | **xx** | **xx** | **xx** | **xx** |  |
| Rx (P) | Negative response | 03 | 7F | 36 | 78 | 00 | 00 | 00 | 00 | Response pending (if necessary) |
| Rx (P) | Positive response | 02 | 76 | nn | 00 | 00 | 00 | 00 | 00 |  |
|  |  |  |  |  |  |  |  |  |  |  |
| **Tx (P)** | **RequestTransferExit (S.F)** | **01** | **37** | **00** | **00** | **00** | **00** | **00** | **00** |  |
| Rx (P) | Negative response | 03 | 7F | 37 | 78 | 00 | 00 | 00 | 00 | Response pending (if necessary) |
| Rx (P) | Positive response | 03 | 77 | xx | xx | 00 | 00 | 00 | 00 | 2-byte CRC |
| : | : |  |  |  |  |  |  |  |  |  |

**Check for Valid Application**

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Tester | Description | Data | | | | | | | | Explanation |
| **Tx (P)** | **RoutineControl 0304H (S.F)** | **04** | **31** | **01** | **03** | **04** | **00** | **00** | **00** | Request check for Valid Application |
| Rx (P) | Negative response | 03 | 7F | 31 | 78 | 00 | 00 | 00 | 00 | Response pending (if necessary) |
| : | : |  |  |  |  |  |  |  |  |  |
| Rx (P) | Negative response | 03 | 7F | 31 | 78 | 00 | 00 | 00 | 00 |  |
| Rx (P) | Positive response | 06 | 71 | 01 | 03 | 04 | 10 | 02 | 00 | Validation Application Software Is Present |

**Exit ECU programming session**

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Tester | Description | Data | | | | | | | | Explanation |
| **Tx (F)** | **ECUReset (S.F)** | **02** | **11** | **81** | **00** | **00** | **00** | **00** | **00** | ECU reset |

* 1. File Upload

The example traces for upload are based upon the flowcharts from chapter 5 File Upload Sequence.

* F = Functional addressing, P = Physical addressing.
* S.F = Single Frame, F.F = First Frame, C.F = Consecutive Frame.
* Zeros in grey are padded, not part of the service.

**Start ECU programming session, identify the ECU and enter security access mode**

| Tester | Description | Data | | | | | | | | Explanation |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Tx (F)** | **DiagnosticSessionControl (S.F)** | **02** | **10** | **82** | **00** | **00** | **00** | **00** | **00** | Start programmingSession |
| **Tx (F)** | **DiagnosticSessionControl (S.F)** | **02** | **10** | **82** | **00** | **00** | **00** | **00** | **00** | Start programmingSession |
| **Tx (F)** | **DiagnosticSessionControl (S.F)** | **02** | **10** | **82** | **00** | **00** | **00** | **00** | **00** | Start programmingSession |
| **Tx (F)** | **DiagnosticSessionControl (S.F)** | **02** | **10** | **82** | **00** | **00** | **00** | **00** | **00** | Start programmingSession |
| **Tx (F)** | **DiagnosticSessionControl (S.F)** | **02** | **10** | **82** | **00** | **00** | **00** | **00** | **00** | Start programmingSession |
| **Tx (F)** | **DiagnosticSessionControl (S.F)** | **02** | **10** | **82** | **00** | **00** | **00** | **00** | **00** | Start programmingSession |
| **Tx (F)** | **DiagnosticSessionControl (S.F)** | **02** | **10** | **82** | **00** | **00** | **00** | **00** | **00** | Start programmingSession |
| **Tx (F)** | **DiagnosticSessionControl (S.F)** | **02** | **10** | **82** | **00** | **00** | **00** | **00** | **00** | Start programmingSession |
| **Tx (F)** | **DiagnosticSessionControl (S.F)** | **02** | **10** | **82** | **00** | **00** | **00** | **00** | **00** | Start programmingSession |
| **Tx (F)** | **DiagnosticSessionControl (S.F)** | **02** | **10** | **82** | **00** | **00** | **00** | **00** | **00** | Start programmingSession |
| **Tx (F)** | **DiagnosticSessionControl (S.F)** | **02** | **10** | **82** | **00** | **00** | **00** | **00** | **00** | Start programmingSession |
| **Tx (F)** | **DiagnosticSessionControl (S.F)** | **02** | **10** | **82** | **00** | **00** | **00** | **00** | **00** | Start programmingSession |
| **Tx (F)** | **DiagnosticSessionControl (S.F)** | **02** | **10** | **82** | **00** | **00** | **00** | **00** | **00** | Start programmingSession |
| **Tx (F)** | **DiagnosticSessionControl (S.F)** | **02** | **10** | **82** | **00** | **00** | **00** | **00** | **00** | Start programmingSession |
| **Tx (F)** | **DiagnosticSessionControl (S.F)** | **02** | **10** | **82** | **00** | **00** | **00** | **00** | **00** | Start programmingSession |
| **Tx (F)** | **DiagnosticSessionControl (S.F)** | **02** | **10** | **82** | **00** | **00** | **00** | **00** | **00** | Start programmingSession |
| **Tx (F)** | **DiagnosticSessionControl (S.F)** | **02** | **10** | **82** | **00** | **00** | **00** | **00** | **00** | Start programmingSession |
| **Tx (F)** | **DiagnosticSessionControl (S.F)** | **02** | **10** | **82** | **00** | **00** | **00** | **00** | **00** | Start programmingSession |
| **Tx (F)** | **DiagnosticSessionControl (S.F)** | **02** | **10** | **82** | **00** | **00** | **00** | **00** | **00** | Start programmingSession |
| **Tx (F)** | **DiagnosticSessionControl (S.F)** | **02** | **10** | **82** | **00** | **00** | **00** | **00** | **00** | Start programmingSession |
|  |  |  |  |  |  |  |  |  |  |  |
| **Tx (P)** | **DiagnosticSessionControl (S.F)** | **02** | **10** | **02** | **00** | **00** | **00** | **00** | **00** |  |
| Rx (P) | Positive response (S.F) | 06 | 50 | 02 | xx | xx | yy | yy | 00 | Application layer timing parameters |
|  |  |  |  |  |  |  |  |  |  |  |
| **Tx (P)** | **ReadDataByIdentifier F111H (S.F)** | **03** | **22** | **F1** | **11** | **00** | **00** | **00** | **00** | Request ECUCoreAssemblyNumber |
| Rx (P) | Positive response (F.F) | 10 | 1B | 62 | F1 | 11 | xx | xx | xx | ECUCoreAssemblyNumber, 24 bytes |
| **Tx (P)** | **Flow control** | **30** | **00** | **00** | **00** | **00** | **00** | **00** | **00** |  |
| Rx (P) | Positive response (C.F) | 21 | xx | xx | xx | xx | xx | xx | xx |  |
| Rx (P) | Positive response (C.F) | 22 | xx | xx | xx | xx | xx | xx | xx |  |
| Rx (P) | Positive response (last C.F) | 23 | xx | xx | xx | xx | xx | xx | xx |  |
|  |  |  |  |  |  |  |  |  |  |  |
| **Tx (P)** | **ReadDataByIdentifier F18CH (S.F)** | **03** | **22** | **F1** | **8C** | **00** | **00** | **00** | **00** | Request ECU Serial Number |
| Rx (P) | Positive response (F.F) | 10 | 13 | 62 | F1 | 8C | xx | xx | xx | ECU Serial Number, 16 bytes |
| **Tx (P)** | **Flow control** | **30** | **00** | **00** | **00** | **00** | **00** | **00** | **00** |  |
| Rx (P) | Positive response (C.F) | 21 | xx | xx | xx | xx | xx | xx | xx |  |
| Rx (P) | Positive response (last C.F) | 22 | xx | xx | xx | xx | xx | xx | 00 |  |
|  |  |  |  |  |  |  |  |  |  |  |
| **Tx (P)** | **SecurityAccess (S.F)** | **02** | **27** | **01** | **00** | **00** | **00** | **00** | **00** | Request seed |
| Rx (P) | Positive response | 05 | 67 | 01 | xx | xx | xx | 00 | 00 | Seed, (e.g., seed length = 3 bytes) |
| **Tx (P)** | **SecurityAccess (S.F)** | **05** | **27** | **02** | **xx** | **xx** | **xx** | **00** | **00** | Key, (e.g., key length = 3 bytes) |
| Rx (P) | Positive response | 02 | 67 | 02 | 00 | 00 | 00 | 00 | 00 | Security access granted |

**Download the SBL**

| Tester | Description | Data | | | | | | | | Explanation |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Tx (P)** | **RequestDownload (F.F)** | **10** | **0B** | **34** | **00** | **44** | **xx** | **xx** | **xx** | Request download, first data block |
| Rx (P) | Flow control | 30 | 00 | 00 | 00 | 00 | 00 | 00 | 00 |  |
| **Tx (P)** | **RequestDownload (C.F)** | **21** | **xx** | **yy** | **yy** | **yy** | **yy** | **00** | **00** |  |
| Rx (P) | Positive response | 04 | 74 | 20 | 0x | xx | 00 | 00 | 00 | maxNumberOfBlockLength |
|  |  |  |  |  |  |  |  |  |  |  |
| **Tx (P)** | **TransferData (F.F)** | **1x** | **xx** | **36** | **01** | **xx** | **xx** | **xx** | **xx** | blockSequenceCounter = 1 |
| Rx (P) | Flow control | 30 | 00 | 00 | 00 | 00 | 00 | 00 | 00 |  |
| **Tx (P)** | **TransferData(C.F)** | **21** | **xx** | **xx** | **xx** | **xx** | **xx** | **xx** | **xx** |  |
|  | : |  |  |  |  |  |  |  |  |  |
| **Tx (P)** | **TransferData(last C.F)** | **2x** | **xx** | **xx** | **xx** | **xx** | **xx** | **xx** | **xx** |  |
| Rx (P) | Positive response | 02 | 76 | 01 | 00 | 00 | 00 | 00 | 00 | blockSequenceCounter = 1 |
| : | : |  |  |  |  |  |  |  |  |  |
| : | : |  |  |  |  |  |  |  |  |  |
| **Tx (P)** | **TransferData (F.F)** | **1x** | **xx** | **36** | **nn** | **xx** | **xx** | **xx** | **xx** |  |
| Rx (P) | Flow control | 30 | 00 | 00 | 00 | 00 | 00 | 00 | 00 |  |
| **Tx (P)** | **TransferData(C.F)** | **21** | **xx** | **xx** | **xx** | **xx** | **xx** | **xx** | **xx** |  |
| : | : |  |  |  |  |  |  |  |  |  |
| **Tx (P)** | **TransferData(last C.F)** | **2x** | **xx** | **xx** | **xx** | **xx** | **xx** | **xx** | **xx** |  |
| Rx (P) | Positive response | 01 | 76 | nn | 00 | 00 | 00 | 00 | 00 |  |
|  |  |  |  |  |  |  |  |  |  |  |
| **Tx (P)** | **RequestTransferExit (S.F)** | **01** | **37** | **00** | **00** | **00** | **00** | **00** | **00** |  |
| Rx (P) | Negative response | 03 | 7F | 37 | 78 | 00 | 00 | 00 | 00 | Response pending (if necessary) |
| Rx (P) | Positive response | 03 | 77 | xx | xx | 00 | 00 | 00 | 00 | 2-byte CRC |
|  |  |  |  |  |  |  |  |  |  |  |
| **Tx (P)** | **RequestDownload (F.F)** | **10** | **0B** | **34** | **00** | **44** | **xx** | **xx** | **xx** | Request download, next data block |
| Rx (P) | Flow control | 30 | 00 | 00 | 00 | 00 | 00 | 00 | 00 |  |
| **Tx (P)** | **RequestDownload (C.F)** | **21** | **xx** | **yy** | **yy** | **yy** | **yy** | **00** | **00** |  |
| Rx (P) | Positive response | 04 | 74 | 20 | 0x | xx | 00 | 00 | 00 | maxNumberOfBlockLength |
|  |  |  |  |  |  |  |  |  |  |  |
| **Tx (P)** | **TransferData (F.F)** | **1x** | **xx** | **36** | **01** | **xx** | **xx** | **xx** | **xx** | blockSequenceCounter = 1 |
| Rx (P) | Flow control | 30 | 00 | 00 | 00 | 00 | 00 | 00 | 00 |  |
| **Tx (P)** | **TransferData(C.F)** | **21** | **xx** | **xx** | **xx** | **xx** | **xx** | **xx** | **xx** |  |
| : | : |  |  |  |  |  |  |  |  |  |
| **Tx (P)** | **TransferData(last C.F)** | **2x** | **xx** | **xx** | **xx** | **xx** | **xx** | **xx** | **xx** |  |
| Rx (P) | Positive response | 02 | 76 | 01 | 00 | 00 | 00 | 00 | 00 | blockSequenceCounter = 1 |
| : | : |  |  |  |  |  |  |  |  |  |
| : | : |  |  |  |  |  |  |  |  |  |
| **Tx (P)** | **TransferData (F.F)** | **1x** | **xx** | **36** | **nn** | **xx** | **xx** | **xx** | **xx** |  |
| Rx (P) | Flow control | 30 | 00 | 00 | 00 | 00 | 00 | 00 | 00 |  |
| **Tx (P)** | **TransferData(C.F)** | **21** | **xx** | **xx** | **xx** | **xx** | **xx** | **xx** | **xx** |  |
| : | : |  |  |  |  |  |  |  |  |  |
| **Tx (P)** | **TransferData(last C.F)** | **2x** | **xx** | **xx** | **xx** | **xx** | **xx** | **xx** | **xx** |  |
| Rx (P) | Positive response | 02 | 76 | nn | 00 | 00 | 00 | 00 | 00 |  |
|  |  |  |  |  |  |  |  |  |  |  |
| **Tx (P)** | **RequestTransferExit (S.F)** | **01** | **37** | **00** | **00** | **00** | **00** | **00** | **00** |  |
| Rx (P) | Negative response | 03 | 7F | 37 | 78 | 00 | 00 | 00 | 00 | Response pending (if necessary) |
| Rx (P) | Positive response | 03 | 77 | xx | xx | 00 | 00 | 00 | 00 | 2-byte CRC |
|  |  |  |  |  |  |  |  |  |  |  |
| **Tx (P)** | **RoutineControl 0301H (F.F)** | **10** | **08** | **31** | **01** | **03** | **01** | **xx** | **xx** | Activate SBL |
| Rx (P) | Flow control | 30 | 00 | 00 | 00 | 00 | 00 | 00 | 00 |  |
| **Tx (P)** | **RoutineControl 0301H (C.F)** | **21** | **xx** | **xx** | **00** | **00** | **00** | **00** | **00** |  |
| Rx (P) | Positive response | 05 | 71 | 01 | 03 | 01 | 10 | 00 | 00 |  |

**Upload data**

| Tester | Description | Data | | | | | | | | Explanation |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Tx (P)** | **RequestUpload (F.F)** | **10** | **0B** | **35** | **00** | **44** | **xx** | **xx** | **xx** | Request upload first data block |
| Rx (P) | Flow control | 30 | 00 | 00 | 00 | 00 | 00 | 00 | 00 |  |
| **Tx (P)** | **RequestUpload (C.F)** | **21** | **xx** | **yy** | **yy** | **yy** | **yy** | **00** | **00** |  |
| Rx (P) | Positive response | 04 | 75 | 20 | 0x | xx | 00 | 00 | 00 | maxNumberOfBlockLength |
|  |  |  |  |  |  |  |  |  |  |  |
| **Tx (P)** | **TransferData (S.F)** | **02** | **36** | **01** | **00** | **00** | **00** | **00** | **00** | blockSequenceCounter = 1 |
| Rx (P) | Positive response (F.F) | 1x | xx | 76 | 01 | xx | xx | xx | xx | blockSequenceCounter = 1 |
| **Tx (P)** | **Flow control** | **30** | **00** | **00** | **00** | **00** | **00** | **00** | **00** |  |
| Rx (P) | Positive response (C.F) | 21 | xx | xx | xx | xx | xx | xx | xx |  |
| : | : |  |  |  |  |  |  |  |  |  |
| Rx (P) | Positive response (last C.F) | 2x | xx | xx | xx | xx | xx | xx | xx |  |
| : | : |  |  |  |  |  |  |  |  |  |
| : | : |  |  |  |  |  |  |  |  |  |
| **Tx (P)** | **TransferData (S.F)** | **02** | **36** | **nn** | **00** | **00** | **00** | **00** | **00** |  |
| Rx (P) | Positive response (F.F) | 1x | xx | 76 | nn | xx | xx | xx | xx |  |
| **Tx (P)** | **Flow control** | **30** | **00** | **00** | **00** | **00** | **00** | **00** | **00** |  |
| Rx (P) | Positive response (C.F) | 21 | xx | xx | xx | xx | xx | xx | xx |  |
| : | : |  |  |  |  |  |  |  |  |  |
| Rx (P) | Positive response (last C.F) | 2x | xx | xx | xx | xx | xx | xx | xx |  |
|  |  |  |  |  |  |  |  |  |  |  |
| **Tx (P)** | **RequestTransferExit (S.F)** | **01** | **37** | **00** | **00** | **00** | **00** | **00** | **00** |  |
| Rx (P) | Negative response | 03 | 7F | 37 | 78 | 00 | 00 | 00 | 00 | Response pending (if necessary) |
| Rx (P) | Positive response | 03 | 77 | xx | xx | 00 | 00 | 00 | 00 | 2-byte CRC |
| : | : |  |  |  |  |  |  |  |  |  |
| **Tx (P)** | **RequestUpload (F.F)** | **10** | **0B** | **35** | **00** | **44** | **xx** | **xx** | **xx** | Request upload, next data block |
| Rx (P) | Flow control | 30 | 00 | 00 | 00 | 00 | 00 | 00 | 00 |  |
| **Tx (P)** | **RequestUpload (C.F)** | **21** | **xx** | **yy** | **yy** | **yy** | **yy** | **00** | **00** |  |
| Rx (P) | Positive response | 04 | 75 | 20 | 0x | xx | 00 | 00 | 00 | maxNumberOfBlockLength |
|  |  |  |  |  |  |  |  |  |  |  |
| **Tx (P)** | **TransferData (S.F)** | **02** | **36** | **01** | **00** | **00** | **00** | **00** | **00** | blockSequenceCounter = 1 |
| Rx (P) | Positive response (F.F) | 1x | xx | 76 | 01 | xx | xx | xx | xx | blockSequenceCounter = 1 |
| **Tx (P)** | **Flow control** | **30** | **00** | **00** | **00** | **00** | **00** | **00** | **00** |  |
| Rx (P) | Positive response (C.F) | 21 | xx | xx | xx | xx | xx | xx | xx |  |
| : | : |  |  |  |  |  |  |  |  |  |
| Rx (P) | Positive response (last C.F) | 2x | xx | xx | xx | xx | xx | xx | xx |  |
| : | : |  |  |  |  |  |  |  |  |  |
| : | : |  |  |  |  |  |  |  |  |  |
| **Tx (P)** | **TransferData (S.F)** | **02** | **36** | **nn** | **00** | **00** | **00** | **00** | **00** |  |
| Rx (P) | Positive response (F.F) | 1x | xx | 76 | nn | xx | xx | xx | xx |  |
| **Tx (P)** | **Flow control** | **30** | **00** | **00** | **00** | **00** | **00** | **00** | **00** |  |
| Rx (P) | Positive response (C.F) | 21 | xx | xx | xx | xx | xx | xx | xx |  |
| : | : |  |  |  |  |  |  |  |  |  |
| Rx (P) | Positive response (last C.F) | 2x | xx | xx | xx | xx | xx | xx | xx |  |
|  |  |  |  |  |  |  |  |  |  |  |
| **Tx (P)** | **RequestTransferExit (S.F)** | **01** | **37** | **00** | **00** | **00** | **00** | **00** | **00** |  |
| Rx (P) | Negative response | 03 | 7F | 37 | 78 | 00 | 00 | 00 | 00 | Response pending (if necessary) |
| Rx (P) | Positive response | 03 | 77 | xx | xx | 00 | 00 | 00 | 00 | 2-byte CRC |
| : | : |  |  |  |  |  |  |  |  |  |

**Exit ECU programming session**

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Tester | Description | Data | | | | | | | | Explanation |
| **Tx (F)** | **ECUReset (S.F)** | **02** | **11** | **81** | **00** | **00** | **00** | **00** | **00** | ECU reset |

1. : Code Examples
   1. CRC16-CITT C-code Example 1 (Fast)

This example uses a look-up table with pre-calculated CRCs for fast calculation.

* fast calculation
* large code size

/\* function declarations \*/

unsigned int CalcCRC(unsigned int size,unsigned char \*data);

/\* test data array \*/

unsigned char data[256] =

{ 0x00,0x01,0x02,0x03,0x04,0x05,0x06,0x07,0x08,0x09,0x0a,0x0b,0x0c,0x0d,0x0e,0x0f,

0x10,0x11,0x12,0x13,0x14,0x15,0x16,0x17,0x18,0x19,0x1a,0x1b,0x1c,0x1d,0x1e,0x1f,

0x20,0x21,0x22,0x23,0x24,0x25,0x26,0x27,0x28,0x29,0x2a,0x2b,0x2c,0x2d,0x2e,0x2f,

0x30,0x31,0x32,0x33,0x34,0x35,0x36,0x37,0x38,0x39,0x3a,0x3b,0x3c,0x3d,0x3e,0x3f,

0x40,0x41,0x42,0x43,0x44,0x45,0x46,0x47,0x48,0x49,0x4a,0x4b,0x4c,0x4d,0x4e,0x4f,

0x50,0x51,0x52,0x53,0x54,0x55,0x56,0x57,0x58,0x59,0x5a,0x5b,0x5c,0x5d,0x5e,0x5f,

0x60,0x61,0x62,0x63,0x64,0x65,0x66,0x67,0x68,0x69,0x6a,0x6b,0x6c,0x6d,0x6e,0x6f,

0x70,0x71,0x72,0x73,0x74,0x75,0x76,0x77,0x78,0x79,0x7a,0x7b,0x7c,0x7d,0x7e,0x7f,

0x80,0x81,0x82,0x83,0x84,0x85,0x86,0x87,0x88,0x89,0x8a,0x8b,0x8c,0x8d,0x8e,0x8f,

0x90,0x91,0x92,0x93,0x94,0x95,0x96,0x97,0x98,0x99,0x9a,0x9b,0x9c,0x9d,0x9e,0x9f,

0xa0,0xa1,0xa2,0xa3,0xa4,0xa5,0xa6,0xa7,0xa8,0xa9,0xaa,0xab,0xac,0xad,0xae,0xaf,

0xb0,0xb1,0xb2,0xb3,0xb4,0xb5,0xb6,0xb7,0xb8,0xb9,0xba,0xbb,0xbc,0xbd,0xbe,0xbf,

0xc0,0xc1,0xc2,0xc3,0xc4,0xc5,0xc6,0xc7,0xc8,0xc9,0xca,0xcb,0xcc,0xcd,0xce,0xcf,

0xd0,0xd1,0xd2,0xd3,0xd4,0xd5,0xd6,0xd7,0xd8,0xd9,0xda,0xdb,0xdc,0xdd,0xde,0xdf,

0xe0,0xe1,0xe2,0xe3,0xe4,0xe5,0xe6,0xe7,0xe8,0xe9,0xea,0xeb,0xec,0xed,0xee,0xef,

0xf0,0xf1,0xf2,0xf3,0xf4,0xf5,0xf6,0xf7,0xf8,0xf9,0xfa,0xfb,0xfc,0xfd,0xfe,0xff

};

unsigned int crctab[256] =

{

0x0000, 0x1021, 0x2042, 0x3063, 0x4084, 0x50A5, 0x60C6, 0x70E7,

0x8108, 0x9129, 0xA14A, 0xB16B, 0xC18C, 0xD1AD, 0xE1CE, 0xF1EF,

0x1231, 0x0210, 0x3273, 0x2252, 0x52B5, 0x4294, 0x72F7, 0x62D6,

0x9339, 0x8318, 0xB37B, 0xA35A, 0xD3BD, 0xC39C, 0xF3FF, 0xE3DE,

0x2462, 0x3443, 0x0420, 0x1401, 0x64E6, 0x74C7, 0x44A4, 0x5485,

0xA56A, 0xB54B, 0x8528, 0x9509, 0xE5EE, 0xF5CF, 0xC5AC, 0xD58D,

0x3653, 0x2672, 0x1611, 0x0630, 0x76D7, 0x66F6, 0x5695, 0x46B4,

0xB75B, 0xA77A, 0x9719, 0x8738, 0xF7DF, 0xE7FE, 0xD79D, 0xC7BC,

0x48C4, 0x58E5, 0x6886, 0x78A7, 0x0840, 0x1861, 0x2802, 0x3823,

0xC9CC, 0xD9ED, 0xE98E, 0xF9AF, 0x8948, 0x9969, 0xA90A, 0xB92B,

0x5AF5, 0x4AD4, 0x7AB7, 0x6A96, 0x1A71, 0x0A50, 0x3A33, 0x2A12,

0xDBFD, 0xCBDC, 0xFBBF, 0xEB9E, 0x9B79, 0x8B58, 0xBB3B, 0xAB1A,

0x6CA6, 0x7C87, 0x4CE4, 0x5CC5, 0x2C22, 0x3C03, 0x0C60, 0x1C41,

0xEDAE, 0xFD8F, 0xCDEC, 0xDDCD, 0xAD2A, 0xBD0B, 0x8D68, 0x9D49,

0x7E97, 0x6EB6, 0x5ED5, 0x4EF4, 0x3E13, 0x2E32, 0x1E51, 0x0E70,

0xFF9F, 0xEFBE, 0xDFDD, 0xCFFC, 0xBF1B, 0xAF3A, 0x9F59, 0x8F78,

0x9188, 0x81A9, 0xB1CA, 0xA1EB, 0xD10C, 0xC12D, 0xF14E, 0xE16F,

0x1080, 0x00A1, 0x30C2, 0x20E3, 0x5004, 0x4025, 0x7046, 0x6067,

0x83B9, 0x9398, 0xA3FB, 0xB3DA, 0xC33D, 0xD31C, 0xE37F, 0xF35E,

0x02B1, 0x1290, 0x22F3, 0x32D2, 0x4235, 0x5214, 0x6277, 0x7256,

0xB5EA, 0xA5CB, 0x95A8, 0x8589, 0xF56E, 0xE54F, 0xD52C, 0xC50D,

0x34E2, 0x24C3, 0x14A0, 0x0481, 0x7466, 0x6447, 0x5424, 0x4405,

0xA7DB, 0xB7FA, 0x8799, 0x97B8, 0xE75F, 0xF77E, 0xC71D, 0xD73C,

0x26D3, 0x36F2, 0x0691, 0x16B0, 0x6657, 0x7676, 0x4615, 0x5634,

0xD94C, 0xC96D, 0xF90E, 0xE92F, 0x99C8, 0x89E9, 0xB98A, 0xA9AB,

0x5844, 0x4865, 0x7806, 0x6827, 0x18C0, 0x08E1, 0x3882, 0x28A3,

0xCB7D, 0xDB5C, 0xEB3F, 0xFB1E, 0x8BF9, 0x9BD8, 0xABBB, 0xBB9A,

0x4A75, 0x5A54, 0x6A37, 0x7A16, 0x0AF1, 0x1AD0, 0x2AB3, 0x3A92,

0xFD2E, 0xED0F, 0xDD6C, 0xCD4D, 0xBDAA, 0xAD8B, 0x9DE8, 0x8DC9,

0x7C26, 0x6C07, 0x5C64, 0x4C45, 0x3CA2, 0x2C83, 0x1CE0, 0x0CC1,

0xEF1F, 0xFF3E, 0xCF5D, 0xDF7C, 0xAF9B, 0xBFBA, 0x8FD9, 0x9FF8,

0x6E17, 0x7E36, 0x4E55, 0x5E74, 0x2E93, 0x3EB2, 0x0ED1, 0x1EF0

};

/\* driver \*/

void main(void)

{

unsigned int crc;

crc=CalcCRC(256,data);

while(1); /\* the result of the above calculation shall be: crc=0x3FBD \*/

}

/\* Calculate CRC \*/

unsigned int CalcCRC(unsigned int size, unsigned char \*data)

{

unsigned int crc=0xffff; /\* initial value \*/

int tmp;

int i;

for(i=0;i<size;i++) {

tmp=(crc>>8)^data[i];

crc=(crc<<8)^crctab[tmp];

}

return crc;

}

* 1. CRC16-CITT C-code Example 2 (Slow)

This example is not using any look-up table, only shift and XOR operations

* small code size
* slow calculation

#define MASK 0x1021 /\* CRC16-CITT mask \*/

/\* function declarations \*/

unsigned int CalcCRC(unsigned int size,unsigned char \*data);

/\* test data array \*/

unsigned char data[256] =

{ 0x00,0x01,0x02,0x03,0x04,0x05,0x06,0x07,0x08,0x09,0x0a,0x0b,0x0c,0x0d,0x0e,0x0f,

0x10,0x11,0x12,0x13,0x14,0x15,0x16,0x17,0x18,0x19,0x1a,0x1b,0x1c,0x1d,0x1e,0x1f,

0x20,0x21,0x22,0x23,0x24,0x25,0x26,0x27,0x28,0x29,0x2a,0x2b,0x2c,0x2d,0x2e,0x2f,

0x30,0x31,0x32,0x33,0x34,0x35,0x36,0x37,0x38,0x39,0x3a,0x3b,0x3c,0x3d,0x3e,0x3f,

0x40,0x41,0x42,0x43,0x44,0x45,0x46,0x47,0x48,0x49,0x4a,0x4b,0x4c,0x4d,0x4e,0x4f,

0x50,0x51,0x52,0x53,0x54,0x55,0x56,0x57,0x58,0x59,0x5a,0x5b,0x5c,0x5d,0x5e,0x5f,

0x60,0x61,0x62,0x63,0x64,0x65,0x66,0x67,0x68,0x69,0x6a,0x6b,0x6c,0x6d,0x6e,0x6f,

0x70,0x71,0x72,0x73,0x74,0x75,0x76,0x77,0x78,0x79,0x7a,0x7b,0x7c,0x7d,0x7e,0x7f,

0x80,0x81,0x82,0x83,0x84,0x85,0x86,0x87,0x88,0x89,0x8a,0x8b,0x8c,0x8d,0x8e,0x8f,

0x90,0x91,0x92,0x93,0x94,0x95,0x96,0x97,0x98,0x99,0x9a,0x9b,0x9c,0x9d,0x9e,0x9f,

0xa0,0xa1,0xa2,0xa3,0xa4,0xa5,0xa6,0xa7,0xa8,0xa9,0xaa,0xab,0xac,0xad,0xae,0xaf,

0xb0,0xb1,0xb2,0xb3,0xb4,0xb5,0xb6,0xb7,0xb8,0xb9,0xba,0xbb,0xbc,0xbd,0xbe,0xbf,

0xc0,0xc1,0xc2,0xc3,0xc4,0xc5,0xc6,0xc7,0xc8,0xc9,0xca,0xcb,0xcc,0xcd,0xce,0xcf,

0xd0,0xd1,0xd2,0xd3,0xd4,0xd5,0xd6,0xd7,0xd8,0xd9,0xda,0xdb,0xdc,0xdd,0xde,0xdf,

0xe0,0xe1,0xe2,0xe3,0xe4,0xe5,0xe6,0xe7,0xe8,0xe9,0xea,0xeb,0xec,0xed,0xee,0xef,

0xf0,0xf1,0xf2,0xf3,0xf4,0xf5,0xf6,0xf7,0xf8,0xf9,0xfa,0xfb,0xfc,0xfd,0xfe,0xff

};

/\* driver \*/

void main(void)

{

unsigned int crc;

crc=CalcCRC(256,data);

while(1); /\* the result of the above calculation shall be: crc=0x3FBD \*/

}

/\* Calculate CRC \*/

unsigned int CalcCRC(unsigned int size, unsigned char \*data)

{

unsigned int crc=0xffff; /\* initial value \*/

int tmp;

int i,j;

for(i=0;i<size;i++){

tmp=data[i]<<8;

for(j=0;j<8;j++){

if((crc ^ tmp) & 0x8000) crc=(crc<<1)^MASK;

else crc<<=1;

tmp<<=1;

}

}

return crc;

}

1. : Example of How to Enter the Bootloader (PBL) From the Application

To fulfil the requirement in [Ford GGDS] it must be possible to enter the bootloader from the application by sending the DiagnosticSessionControl(programmingSession) service only once (although periodic transmission as described in section 4.1 is recommended for the normal download process).

To avoid problems with hardware initialization (i.e., write once registers, PLL-setup, memory re-map, interrupt vector table re-map, etc.) the preferred solution is that the bootloader always starts executing from reset when the application SW receives a DiagnosticSessionControl(programmingSession) service. The flow charts below describe a method for exiting the application SW and entering the bootloader.

For a faulty programmed ECU or an aborted programming, the "backdoor solution" with a 20 ms time window for receiving DiagnosticSessionControl(programmingSession) after reset can still be used.



Figure 11.1 Preparation by the application before entering the bootloader



Figure 11.2 Bootloader program flow

1. : Conversion of All or Part of an ECU From Programmable to Masked ROM

This appendix has been obsoleted and is no longer relevant.

1. : ECU Programming Possibilities and Identification Overview

This appendix is intended as a high-level overview of the various scenarios that may be encountered for programmable ECUs. This section is not intended to contain all necessary details / requirements for implementation, and any such additional bootloader resources and interfaces between gateways and subnetwork ECUs must be defined outside of this specification. The software identification DIDs described in this appendix are only applicable if the ECU releases its software through WERS.

* 1. ECUs on Main Public Network
     1. Implementation Details

This is the normal and typical use case.

* + 1. Identification Details

The tester is always able to identify the hardware and all non-programmable software by reading DID F111H (ECU Core Assembly Number) in all sessions from the ECU being programmed.

The tester is always able to identify each separately programmable software file by reading one of the standard software identification DIDs from the ECU being programmed after a valid application is present and while a non-programming Session is active. The aforementioned standard software identification DIDs are defined in Table 11.1.

Table 11.1 – Standard Software Identification DIDs

|  |  |  |
| --- | --- | --- |
| DID (Hex) | Name | Description |
| F188 | Vehicle Manufacturer ECU Software Number | WERS part number for the ECU main software |
| F120 | ECU Software #2 Part Number | WERS part number for a separate ECU software part that is not the ECU main software. The part number identifies an ECU software part that consists, at least partly, of a code sequence that is directly executable by the ECU micro-controller. |
| F121 | ECU Software #3 Part Number | See ECU Software #2 Part Number |
| F122 | ECU Software #4 Part Number | See ECU Software #2 Part Number |
| F123 | ECU Software #5 Part Number | See ECU Software #2 Part Number |
| F124 | ECU Main Calibration Data Number | WERS part number for ECU main calibration data. The part number identifies software that consists of any type of data and no code sequence that is directly executable by the ECU micro-controller. For example it can be calibration and/or configuration data and/or a description of an algorithm. |
| F125 | ECU Calibration Data #2 Number | WERS part number for a separate ECU calibration data part that is not the ECU Main Calibration Data. The part number identifies software that consists of any type of data and no code sequence that is directly executable by the ECU micro-controller. For example it can be calibration and/or configuration data and/or a description of an algorithm. |
| F126 | ECU Calibration Data #3 Number | See ECU Calibration Data #2 Number |
| F127 | ECU Calibration Data #4 Number | See ECU Calibration Data #2 Number |
| F128 | ECU Calibration Data #5 Number | See ECU Calibration Data #2 Number |
| F108 | ECU Network Signal Calibration Number | A WERS part number for the network signal file downloaded to an ECU (if this is separately downloadable). This file contains the vehicle signal database information but it also contains data specific to each ECU (e.g., signals used by the ECU, timing for publishing the signals etc.). Because the file is different for each ECU the part number is also different between ECUs (even though the data source potentially comes from the same vehicle message list as other ECUs). |
| F10A | ECU Cal-Config Part Number | WERS part number used for a separate ECU calibration data part that does **not** affect the end item ECU Assembly Number. The part number identifies software that consists of any type of data, but primarily used when containing no code sequence that is directly executable by the ECU micro-controller. |
| F16B | ECU Cal-Config #2 Part Number | WERS part number used if multiple ECU Cal-Config Part Number DIDs are required. See ECU Cal-Config Part Number. |
| F16C | ECU Cal-Config #3 Part Number | See ECU Cal-Config #2 Part Number |
| F16D | ECU Cal-Config #4 Part Number | See ECU Cal-Config #2 Part Number |
| F16E | ECU Cal-Config #5 Part Number | See ECU Cal-Config #2 Part Number |
| F17D | ECU Cal-Config #6 Part Number | See ECU Cal-Config #2 Part Number |
| F10E | ECU Cal-Config #7 Part Number | See ECU Cal-Config #2 Part Number |

* 1. ECUs on Transparent CAN Sub-Network
     1. Implementation Details

Refer to [Ford GGDS] for implementation details and requirements for CAN transparent sub-networks. With this approach, the goal for ECU programming is that the tester will program ECUs on the transparent CAN sub-network no differently than ECUs on the main public network. In other words, the tester will not need to be cognizant that the ECU being programmed is not truly on the main public network.

* + 1. Identification Details

The identification details are the same as for E.1. The tester is always able to identify the hardware and all non-programmable software by reading DID F111H (ECU Core Assembly Number) in all sessions. The tester is always able to identify each separately programmable software file by reading one of the standard software identification DIDs (see E.1.2) from the ECU being programmed after a valid application is present and while a non-programming Session is active.

**NOTE**: SWDL files are released as relevant for the transparent sub-node, with no relationship to the gateway.

* + 1. Sub Node Supports Diagnostics for Programming Only

It is possible that the ECU on the transparent CAN sub-network only supports diagnostics through the gateway for the purposes of programming and that another ECU supports any other diagnostics (e.g., storing of DTCs) for the sub-node. If this case is supported, this means the ECU on the transparent CAN sub-network must still support diagnostics in the application, but may only support a very limited subset of diagnostic services in the application including only the defaultSession. At minimum, the ECU application on the transparent CAN sub-network shall be able to receive and process diagnostic services 10H subfunctions 01H and 02H (i.e., change diagnostic session at least between default and programming), service 11H subfunction 01H (ECU reset), service 3EH (tester Present) and service 22H support of all necessary identification DIDs in appropriate sessions (this includes core assembly, software part numbers, and DID D100H). This is required for the tester to be able to change diagnostic sessions for identifying the ECU no differently than it would if the ECU supported full diagnostics.

As an example, the actual network architecture implementation may be:

Main Public Network (e.g., MS-CAN)

Transparent CAN Sub-Network

ECU C (Transparent Diagnostic Gateway)

ECU B

[ISO 15031-3] Connector

Tester

ECU A

ECU D

ECU E

Transparent

Sub Nodes

From the tester's perspective, it communicates no differently than if the network architecture were:

Main Public Network (e.g., MS-CAN)

ECU C (Transparent Diagnostic Gateway)

ECU B

[ISO 15031-3] Connector

Tester

ECU A

ECU D

ECU E

* 1. Sub Micros within ECU on a Public Main Network
     1. Implementation Details

This use case covers the scenarios when there are multiple micros within a given ECU assembly that may require programming. With this use case, any changes to the hardware of any of the micros contained within the ECU assembly shall always result in an update to the reported ECU Core Assembly Number (i.e., DID F111H).

In order for the main micro to program one of the sub micros, a "memory mapped" address range is utilized. For example, if the valid programmable memory areas of the main micro are 00000000H – 00FFFFFFH, then an address range of 01000000H – 01FFFFFFH could be used to signify that the data is for the sub micro #1 and an address range of 02000000H – 02FFFFFFH could be used to signify that the data is for the sub micro #2.

Using this approach, software files for a sub micro are simply released as an additional software or calibration file that is downloaded to the ECU assembly. In other words, the tester downloads the file no differently than it downloads the main micro's strategy or calibration file. With this scenario, the onus is on the main micro ECU to recognize that the address to erase / program is intended for a sub micro and automatically do any translation and internal communication to the sub micro to ensure that the requested diagnostic functionality is performed and that an appropriate response is returned to the tester. When implemented using this approach, the details of the internal communication shall be transparent to the tester so that it is not cognizant that the data being downloaded is for the main micro or a sub micro. Note that when using this "memory mapped" approach it is possible for the data for the sub micro(s) to be in the same software file as the data of the main micro.

* + 1. Identification Details

The identification details are the same as for E.1. The tester is always able to identify the hardware and all non-programmable software by reading DID F111H (ECU Core Assembly Number) in all sessions. The tester is always able to identify each separately programmable software file by reading one of the standard software identification DIDs (see E.1.2) from the ECU being programmed after a valid application is present and while a non-programming Session is active.

**NOTE**: SWDL files are released as relevant for the ECU assembly and the various strategy or calibration files appear to a tester no differently than they would if a single micro existed and had multiple strategy files or calibration files that are released separately.

As an example, the actual network architecture implementation may be:

Tester

ISO 15031-3 Connector

Main Micro (ECU A)

Core Assembly P/N (F111)

Strategy P/N (F188)

Calibration P/N (F124)

Sub Micro #1

Strategy P/N (F120)

Calibration P/N (F125)

From the tester's perspective, it communicates no differently than if the network architecture were:

Tester

ISO 15031-3 Connector

Main Micro (ECU A)

Core Assembly P/N (F111)

Strategy 1 P/N (F188) Strategy 2 P/N (F120)

Calibration P/N (F124) Calibration 2 P/N (F125)

* 1. ECUs on Public Sub-Network Connected Via Explicit Gateway
     1. Implementation Details

With this use case, the tester must be cognizant that the ECU being programmed is on a public sub-network. In this scenario, the tool must explicitly put the interface ECU which connects the main public network and the sub-network into the gateway state (e.g., Routine 0300H). The tester explicitly puts the gateway ECU into a gateway state to point at the ECU which is to be programmed. After this, the tester programs the ECU by sending the requests to the interface ECU, which then performs any translation and communication necessary on the sub-network.

With this approach, the details of the sub-network itself are transparent to the tester. In other words, the tester does not need to know the sub-network protocol (i.e., whether it is CAN, LIN, or anything else). This approach could also be used to support an architecture similar to that described in E.3 (i.e., sub-nodes are actually micros in the same housing as the gateway ECU).

**NOTE**: SWDL files released for the ECU in this scenario must indicate that the ECU is on a sub-network and must contain the physical diagnostic address of the interface ECU.

The ECU which is to be programmed is responsible for implementing the bootloader requirements (e.g., programmingSession and pseudo defaultSession) in a manner compliant to the SWDL specification. In addition, the ECU being programmed is responsible for implementing diagnostics in the application as described in [Ford GGDS] (with the possible exception as described in E.4.3).

* + 1. Identification Details

The tester is first required to place the interface ECU on the main public network into explicit gateway state and direct communication to the desired sub-node. Once the explicit gateway state is established, the tester will identify the ECU on the public sub-network no differently than identifying an ECU on the main public network.

While explicit gateway state is active, identifying the ECU is the same as for E.1. The tester is always able to identify the hardware and all non-programmable software by reading DID F111H (ECU Core Assembly Number) in all sessions. The tester is always able to identify each separately programmable software file by reading one of the standard software identification DIDs (see E.1.2) from the ECU being programmed after a valid application is present and while a non-programming Session is active.

* + 1. Sub Node Supports Diagnostics for Programming Only

It is possible that the ECU on the public sub-network only supports diagnostics through the gateway for the purposes of programming and that another ECU supports any other diagnostics (e.g., storing of DTCs) for the sub-node. If this case is supported, this means the ECU on the public sub-network must still support diagnostics in the application, but may only support a very limited subset of diagnostic services in the application including only the defaultSession. At minimum, the ECU application on the public sub-network shall be able to receive and process diagnostic services 10H subfunctions 01H and 02H (i.e., change diagnostic session at least between default and programming), service 11H subfunction 01H (ECU reset), service 3EH (tester Present) and service 22H support of all necessary identification DIDs in appropriate sessions (this includes core assembly, software part numbers, and DID D100H). This is required for the tester to be able to change diagnostic sessions for identifying the ECU no differently than it would if the ECU supported full diagnostics.

As an example, the actual network architecture implementation may be:

Public sub-network

(any protocol)

Tester

ISO 15031-3 Connector

ECU A (Gateway)

Core Assy P/N (F111)

Strategy P/N (F188)

Calibration P/N (F124)

Public Sub-node 1

Core Assy P/N (F111)

Strategy P/N (F188)

Calibration P/N (F124)

Public Sub-node 2

Core Assy P/N (F111)

Strategy P/N (F188)

Calibration P/N (F124)

HS-CAN

* 1. ECUs on Private Sub-Network (Not a Valid Implementation)
     1. Implementation Details

This has been deemed as **NOT** a valid use case for performing programming of ECUs by Ford tools. The definition of a private network is one that does not support for any gatewaying of diagnostic requests/responses from a tester. The private sub-node identification DIDs in [Ford GMRDB] are there to support identification of the sub-nodes from the gateway to ensure correct installation, etc. The explicit gateway approach shall be used as described in E.4 to support programming of this architecture.

The remaining text in this section is kept only as documentation of what has been discussed regarding this approach and is not valid for implementation.

*With this use case, there is no explicit diagnostic gateway support for communication to the ECU. In this scenario, the implementation would be most similar to section E.3. The difference in this case is that the core assembly information of the ECU being programmed is NOT contained within the housing of the interface ECU assembly and therefore a change to the hardware of the ECU on the private sub-network is not reflected in an update to the reported ECU Core Assembly number (i.e., DID F111H) of the interface ECU. From this perspective, a tester must be cognizant that the ECU is on a private sub-network as there are different DIDs required to identify the ECU.*

*From a pure programming perspective, the file to be programmed would be released as an additional software or calibration file that is downloaded to the interface ECU assembly (i.e., the same as in section E.3). A "memory mapped" address range would need to be utilized so that the tester downloads the file no differently than it downloads the interface ECUs strategy or calibration file. With this scenario, the onus is still on the interface ECU to recognize that the address to erase / program is intended for an ECU on a private sub-network and automatically do any translation and communication to the ECU on the private sub-network to ensure that the requested diagnostic functionality is performed and that an appropriate response is returned to the tester. When implemented using this approach, the details of the sub-network shall be completely transparent to the tester. In other words, the tester does not need to know the sub-network protocol (i.e., whether it be CAN, LIN, or anything else).*

* + 1. Identification Details

The remaining text in this section is kept only as documentation of what has been discussed regarding this approach and is not valid for implementation.

*The tester is always able to identify the hardware and all non-programmable software by reading one of the DIDs in the range of F129H - F140H (Private Sub Node #1 Core Assembly Number – Private Sub Node #24 Core Assembly Number). ECUs connected as a private sub-node are only capable of having a single software part downloaded. The tester is always able to identify the programmable software file by reading one of the DIDs in the range of F0E8H - F0FFH (Private Sub Node #1 Software Number – Private Sub Node #24 Software Number).*

***NOTE****: SWDL files are released as relevant for the ECU assembly and the various strategy or calibration files appear to a tester no differently than they would if a single micro existed and had multiple strategy files or calibration files that are released separately. However, they are reported out very differently*.

As an example, the actual network architecture implementation may be:

HS-CAN

Private sub-network   
(any protocol)

Tester

ISO 15031-3 Connector

ECU A

Core Assy P/N (F111)

Strategy P/N (F188)

Calibration P/N (F124)

Private Sub-node 1

Core Assy P/N (F129)

Strategy P/N (F0E8)

Public Sub-node 2

Core Assy P/N (F12A)

Strategy P/N (F0E9)

From the tester's perspective, it communicates no differently than if the network architecture were:

HS-CAN

ECU A

Core Assy P/N (F111)

Strategy P/N (F188)

Calibration P/N (F124)

Private SubNode 1 Core Assy P/N (F129)

Private SubNode 1 Software P/N (F0E8)

Private SubNode 2 Core Assy P/N (F12A)

Private SubNode 2 Software P/N (F0E9)

Tester

ISO 15031-3 Connector

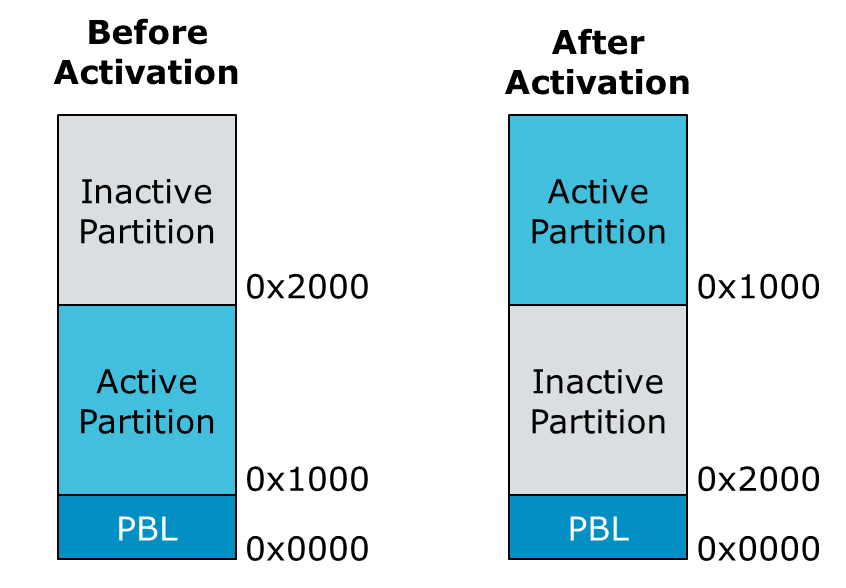
* + 1. Issues with Approach

One of the primary issues with this approach is that Ford IVS is not architected or designed to capture the scenario where there are separate physical assemblies (i.e., private sub-nodes), yet their information (including core assembly information) is reported via unique DIDs from a gateway module. Supporting the scenario would require significant changes to allow capturing of the vehicle architecture as well as any and all relationships and dependencies between hardware and software. FCSD would therefore not have a mechanism to correctly program these ECUs and associated software and core assembly identification information with the correct sub-node, etc. The explicit gateway approach described within this specification will meet the needs and do so in a way consistent with all Ford processes (e.g., IVS processes, NetCom processes, FCSD processes, etc.).

1. : Inactive Memory Architecture Types

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

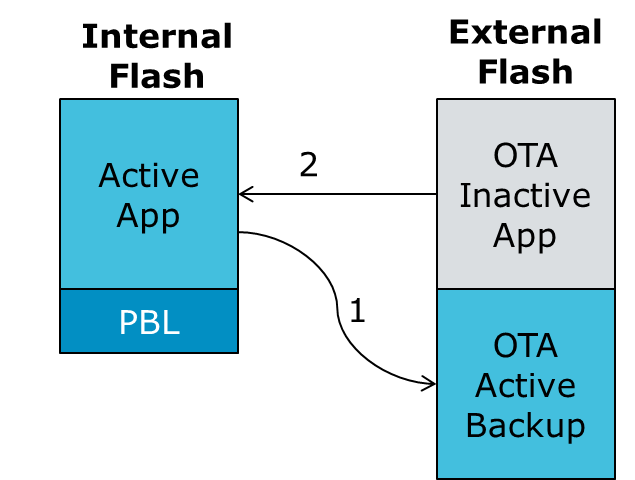
* 1. Inactive Memory Architecture Type 1 – Hardware Facilitated Address Remapping

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
  1. Inactive Memory Architecture Type 2 –Memory Caching Option 1

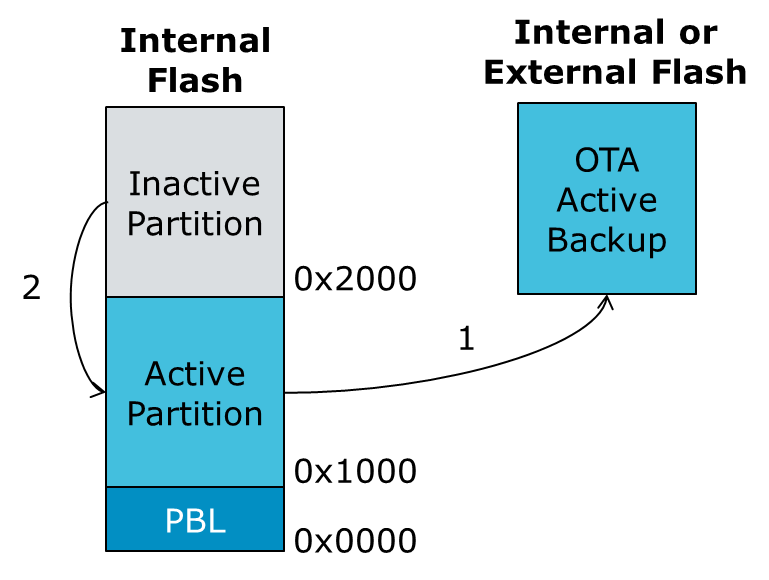
With this approach, the new software is downloaded into an allocated external memory area. Prior to activation of the new software, the currently active application is backed up into external memory and the new software is then copied into the active internal memory by the bootloader.



High Level Requirements:

* 2x external flash to support storage of both A & B memory
  1. Inactive Memory Architecture Type 3 – Memory Caching Option 2

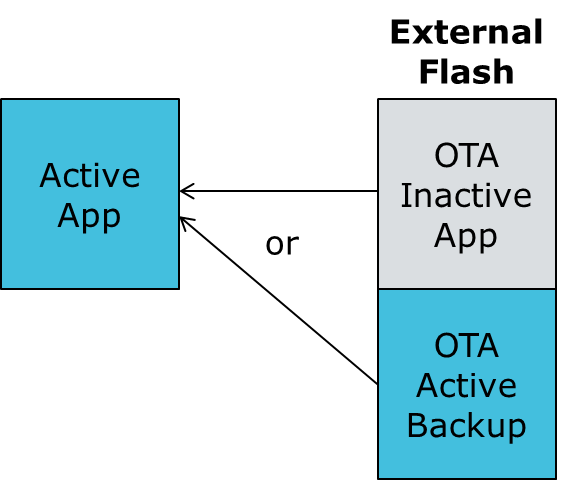
With this approach, the new software is downloaded into an allocated internal memory area. Prior to activation of the new software, the currently active application is backed up into a dedicated backup location in either internal or external memory and the new software is then copied from the inactive 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
  1. Inactive Memory 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