企 业 技 术 标 准

Flash Reprogramming Specification

（Revision: 2.3）

2013-06-20 发布 2013-06-25 实施

广汽集团汽车工程研究院 发布

Content

1 Document Information

1.1 History

Author	Date	Version	Remarks
Huanglifang	2009-02-25	0.50	Initial Draft
Huanglifang	2009-04-01	0.60 (0.91)	Added WDBI request format
Huanglifang	2009-04-20	0.70 (0.92)	Modified chap. 6.1 Diag Session Management Modified CommunicationControl service Modified chap. 6.2.2 Added chap. 6.3.1.1
Huanglifang	2009-04-27	1.0	Modified figure 6-3 Modified figure 6-1 Changed fingerprint DID to $F1 $84 Renamed RoutineControl – routines Additional minor enhancements
Huanglifang	2009-12-25	1.01	Modified figure 6-3 Added extendedSession request in figure 6-4 Minor corrections
Huanglifang	2010-03-01	2.0	Modified figure 6-3 and 6-4 Modified Table 7-3 Changed Routine ID FF00 to $02$02 in section 6.2.2.4
Huanglifang	2010-10-20	2.1	Monified chap. 4.6.2: Details for CRC32 algorithm; Updated information of section 6.2.1; Removed section 6.2.3.3-6.2.3.5; Added WriteDataByIdentifier $2E $xx $yy in step3 of reprogramming steps; *Added* *(R3210);* Added Table 7-4 segment Table and segment Table related description*(R4212)* ;
Huanglifang	2012-02-09	2.2	Added”flash driver need not “segment table” in *R4212*; Added”including ”segment table” bytes in *R4620*

			Added sentence"The memory address is "00 00 00 00" in the $34 service request when download flash driver" in section 6.2.2.4 6.3.3.6, added the detailed information of DID F199; Added R3211.
Huanglifang	2013-02-01	2.3	1.Moved table 7-4 to table 3-2, and updated the contents of table 3-2; 2.Monified the contents of R4212 in section 4.2.5; 3.Added sentence"The whole logical block is earased in one time, including the spare spaces among the segments" in section 3.5.2; 4.Added sentence"While the counter of programming attempts will increase by 1 after positive response of the service request WriteDataByIdentifier-WriteFingerprint" in section 3.5.5; 5.Added section 3.5.6; 6.Added Table 4.4 in section 4.4; 7.Added "the downloaded data bytes not including "segment table" bytes in section 4.6.2; 8. Changed the seguence of figure 6-2; 9. Monified some contents of section 6.2.1.5 and 6.2.2.4

Table 1-1 History of the Document

1.2 Reference Documents

No.	Source	Title	Version
[1]	GAC	GAC Diagnostic Specification 1.01	1.0.1
[2]	ISO	ISO14229-1 Road vehicles - Unified Diagnostic Services (UDS) Part 1	
[3]	ISO	ISO 15765-2: Road vehicles – Diagnostics on CAN – Part 2	
[4]	GAC	GAC internal document – CAN ID assignment	

Table 1-2 Reference Documents

1.3 Document Conventions

In this document, the following terminology applies:

- „Shall" expresses an obligatory/mandatory requirement.
- „Should" expresses a recommendation or an advice.
- „Must" expresses a legal or normative requirement.
- "Will" expresses a precautionary consideration or an additional / optional feature.
- „May" expresses a permitted practice / method, not to be considered as a requirement.

2 Introduction

2.1 Scope of this Document

This document specifies the requirements for flash reprogramming based on UDS. A reprogramming process including standardized interfaces[1] and communication sequences is needed to be able to support ECU software updates during the development phase and in-service of automotive ECUs. This specification provides the requirements for reprogramming, the architecture and the download process flow to support flash reprogramming of an ECU.
This document focuses on CAN based systems only. It does not replace the existing normative documents regarding diagnostics, but it adds additional requirements and restrictions to the contents of the standards. In case of conflicts this document takes priority over the normative documents.
The content of this specification is mandatory for all reprogrammable and non-reprogrammable ECUs. Any deviation requires the approval of GAC and shall be documented within the corresponding ECU diagnostic specification.

2.2 Target Group

This document is intended for development engineers, suppliers of diagnostic equipment and ECU suppliers who are engaged in reprogramming and software updating of ECUs and those who have to equip their ECU with flash reprogramming capabilities.

2.3 Fundamentals

A reprogrammable ECU contains two executable software packages, the real ECU application and the bootloader software. During normal ECU operation, the application software is executed. The bootloader software is active only if it is started by the application in order to download a modified version of the application, or if no application is available on the system.

---

[1] Interfaces e.g. of the communication layers like CAN, Transport Layer, Diagnostics Layer and their parameters in order to assure proper communication between tester and ECU.

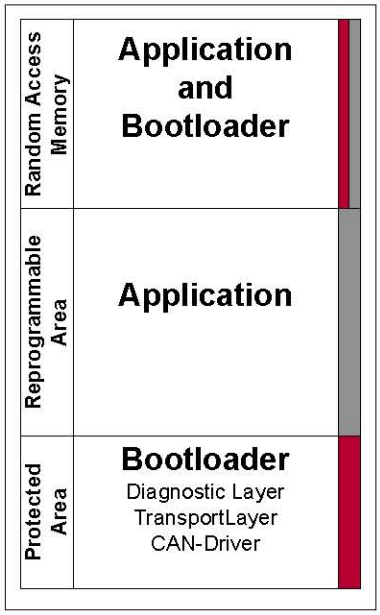

Figure 2-1 Symbolic memory map of a reprogrammable ECU

Application and bootloader occupy a dedicated area of flash memory as depicted in the figure above. Since either the bootloader or the application is executed, the RAM memory of the system can be fully used by both software packages.
The bootloader software uses the diagnostic services of UDS as protocol for download communication. Therefore, the bootloader has got a communication stack of CAN driver, transport layer and a subset of the UDS protocol layer.

3 Flash EEPROM Reprogramming Requirements

3.1 Requirements for non-reprogrammable ECUs

(R3100) In order to operate reprogrammable and non-reprogrammable ECUs together in one network, non-reprogrammable ECUs shall support the diagnostic services specified in 6.3.2 except the DiagnosticSessionControl –programmingSession request.
(R3101) If a non-reprogammable ECU receives a DiagnosticSessionControl – programmingSession ($10 $02) it shall send the negative response code $12 (subFunctionNotSupported).

3.2 General Requirements

(R3200) The system shall be able to enter the bootloader on request from normal operating mode or if no ECU application software is available.
(R3201) The system shall be reprogrammable after a failed or interrupted reprogramming attempt, after a timeout or a reset occurred, and with partly erased flash memory or invalid application.
(R3202) The system shall execute boot code if the application code is missing, invalid or corrupt.
(R3203) A system executing boot code must not disturb normal communication on the network.
(R3204) All ECUs connected to the network shall disable their normal message transmission when they are not reprogrammed and after they received a disable normal message transmission request. Normal communication shall be enabled on all ECUs after an explicit request to enable normal communication, after return to the default session and after a PowerOn/Reset.
(R3205) All ECUs connected to the network shall disable their DTC settings after they received a request to disable fault code setting. Fault code setting shall be enabled on all ECUs after an explicit request to enable fault code setting, after return to the default session and after a PowerOn/Reset.
(R3206) Flash reprogramming must not be started before a complete set of fingerprint data has been transferred to the ECU.
(R3207) After all download data has been transferred, the integrity of the programmed data shall be ensured by a verification routine. Therefore, an explicit check value shall be provided to be compared against the value calculated on the ECU. For further information, refer to 4.6.2.
(R3208) If a watchdog is used on the ECU, it shall be served during the complete flash reprogramming sequence.
(R3209) A programming dependency check shall be performed after download if more than one module can be reprogrammed on the system. For further information, refer to 6.2.2.8 and 6.3.3.7.4 – checkProgrammingDependencies.

(R3210) The boot software only can be reprogrammed by supplier, while the application software and calibration files can be reprogrammed at assembly plant or service.
Supplier should send the application software and calibration files to GAC in Bin file format.
(R3211) Bin file is a binary file which is comprised of header and data bytes. The definition of header is specified in table 3-2. The data bytes have no specific regulation.

Offset	Length (byte)	Name	Description
$0	3	DCID	Data Compatibility Identifier Value This field is used to check the programming dependencies. For an application software block file header this value is compared against a constant in the bootloader to ensure that the software being downloaded is compatible with the boot software in the ECU. For an application data block file header this value is compare by the programming executive against a corresponding value in the application software header to ensure that there is no mismatch between the application data file and the application software. The detailed value of DCID is defined by GAC.
$3	3	SWV	SoftWare Version Version number of SW (S.A,S.B...S.Z)
$6	14	SWPN	SoftWare Part Number Part number of SW(The first 14 bytes of DID F189)
$14	3	AWV/ CWV	Application Version or Calibration vertion Version number of Application/Calibration SW (A.A,A.B...A.Z/ C.A,C.B...C.Z)
$17	9	System name	ECU code and Suppiler Identification
$20	1	NOAR	Number Of Segments This field is used to accommodate programming of a application software or application data Logic block that has data that is broken up into several segments of separate memory regions in the ECU. The number in this field represents the number of different address regions addressed when the block is downloaded.
$21	4	AR1	Address of First Segment
$25	4	LR1	Length of First Segment
$29	4	AR2	Address of Second Segment
$2D	4	LR2	Length of Second Segment
$31	4	AR3	Address of Third Segment
$35	4	LR3	Length of Third Segment

…	…	…	…

Table 3-2 Segment Table

3.3 Resource Requirements

(R3300) The flash memory consumption of the bootloader must be kept to minimum since the available flash memory of an ECU is shared by application and bootloader. The available ECU RAM memory can be fully occupied by the bootloader and is also fully available for the application.
(R3301) In order to reduce power consumption, the ECU shall enter its sleep mode when it executes boot code in idle mode (Default Session) for more than 300 seconds.

3.4 Security Requirements

(R3400) Before a reprogramming procedure is started, the security access service shall be passed successfully.
(R3401) Critical parts of the flash driver code must not be stored on the ECU, but shall be downloaded during the reprogramming procedure (software interlock). This ensures that the flash programming routines (erasing and writing) cannot be activated accidentally during normal operating mode.
(R3402) The boot software shall be stored in a protected memory area (software and/or hardware protection).

3.5 Download Of Flash Data

3.5.1 Addressing

(R3501) The addressing scheme for download data is based on a linear address space. Depending on the requirements of the underlying microcontroller platform, linear addresses must be transformed into physical addresses by the bootloader whenever flash memory is accessed for reading, writing and erasing. The conversion of linear addresses into physical ones shall be done individually according to the hardware-specific conventions. It is also possible that no conversion is necessary. That way, it is possible to support banked flash memory devices, EEPROM- and external flash memory devices in one ECU.

3.5.2 Erasing Of Flash Memory

(R3502) Flash devices require that flash memory is erased before it is reprogrammed. Any flash cell must not be reprogrammed without a previous erase procedure in order to achieve the full data retention time of the device.

The whole logical block is earased in one time, including the spare spaces among the segments.

3.5.3 Ascending Address Order

(R3503) All download data must be transferred to the ECU in ascending address order as depicted in Figure 4-3. This requirement makes sure that there is no multiple subsequent write access to flash cells without erasing.

3.5.4 Flash Programming Conditions

(R3504) Before the flash reprogramming procedure is started, programming conditions must be checked. The ECU that shall be reprogrammed might be in a condition that does not permit ECU reprogramming. In this case, a reprogramming attempt must be rejected. The check for flash programming conditions shall be performed when the ECU receives the RoutineControl – “checkProgrammingPreconditions” service request.

3.5.5 Programming Attempt Counter

There may be some reasons to limit the number of programming attempts for an ECU, e.g. because a flash device only permits a certain number of erase- and programming cycles.
(R3510) The bootloader software shall count the number of programming attempts for each logical block.
(R3511) Before a fingerprint is accepted in the WriteDataByIdentifier ($2E $F1 $84) service implementation, the programming attempt counter of the requested logical block shall be checked. If the programming attempt counter exceeds a predefined value, a reprogramming attempt must be denied by sending a negative response upon the service request WriteDataByIdentifier-WriteFingerprint.
While the counter of programming attempts will increase by 1 after positive response of the service request WriteDataByIdentifier-WriteFingerprint.
(R3512) The maximum number of reprogramming attempts depends on the applied flash device and the limitations defined by the OEM and shall be configured for each ECU individually.
(R3513) If the maximum number of programming attempts is set to the value of 0, programming attempts are unlimited.

3.5.6 Programming Counter

The bootloader software shall count the number of successful programming times .
The counter of programming will increase by 1 after positive response of the service request CheckProgrammingDependencies.

.

4 Architectural Requirements

4.1 Functional Overview

The bootloader software is subdivided into several functional blocks as depicted in Figure 4-1. It is stored in the protected memory area of the ECU so that the bootloader can be activated at any time. The ECU can be reprogrammed despite of potential error conditions.

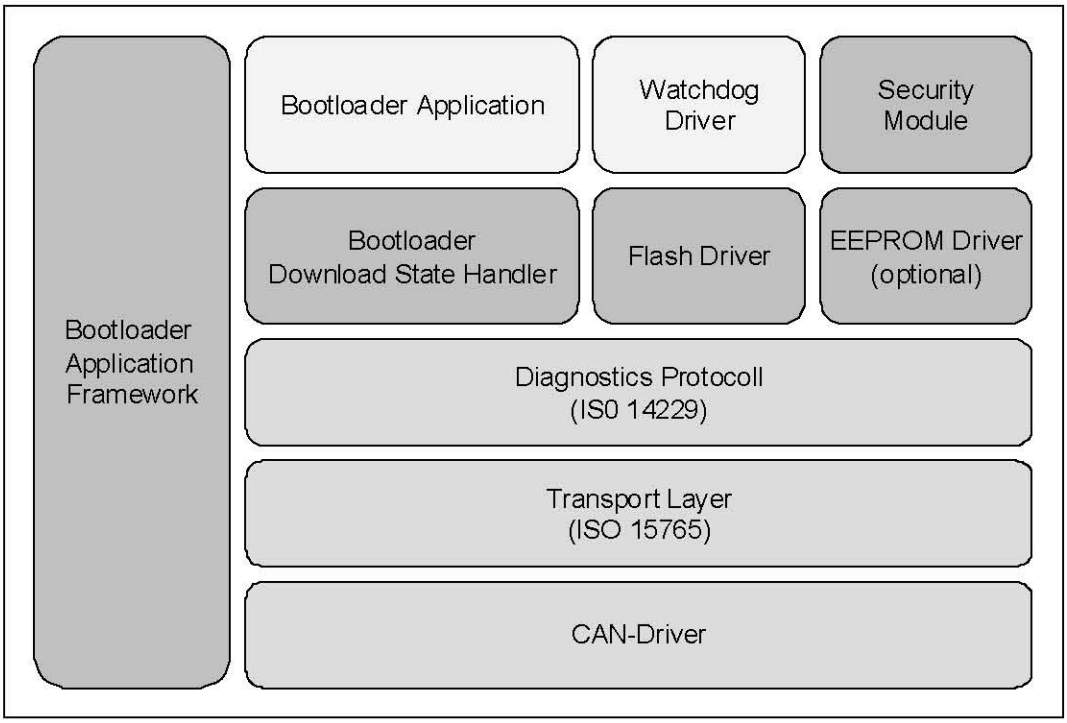

Figure 4-1 Functional blocks of the bootloader

The functionality of the shown blocks is described in the table below.

Functional Block	Description
Bootloader Application Framework	The bootloader application framework provides the boot procedure and the time base handling for the bootloader components
Bootloader Application	This module permits supplier-specific code in order to support system specific implementations.
Watchdog Driver	The watchdog driver implements an ECU-specific code to serve the watchdog(s) of the system
Security Module	The security module provides implementations required for the security access service and the verification of the download.

Bootloader Download State Handler	This component of the bootloader is closely related to the diagnostic layer and controls the states of the download sequence and the flash programming
Flash Driver	The flash driver provides routines for flash erasing and programming.
EEPROM-Driver	The EEPROM driver provides routines for EEPROM erasing and programming.
Diagnostic Protocol	This module provides all diagnostic services needed for the download procedure.
Transport Protocol	ISO-15765-2 -compliant transport layer to support segmented transmission of diagnosis/ download data.
CAN-Driver	Hardware specific CAN driver

Table 4-1 Description of functional blocks

4.2 Flash Memory Partitioning

4.2.1 Physical Flash Sectors

(R4200) The bootloader shall take into account that the underlying flash technology determines the minimum number of bytes that must be erased all at once. This size is called a physical flash sector. The memory area of a flash device usually is divided into several flash sectors.

4.2.2 Physical Flash Pages

(R4201) The bootloader shall take into account that the underlying flash technology determines the minimum number of bytes that must be programmed all at once. This size is called a physical flash page.

4.2.3 Logical Blocks

The flash memory space of an ECU can be partitioned into logical blocks. One logical block can be reprogrammed independently from other blocks. The basic motivation for flash partitioning is to avoid reprogramming of the complete flash memory when only a certain part of the application software or application data has changed.

(R4202) The partition that is occupied by the bootloader is called boot block and cannot be reprogrammed. The contents of a reprogrammable logical block can be determined by the ECU supplier and can consist of code, data or both. The supplier can decide if and how the application code and data is subdivided into units using logical blocks.

(R4203) Logical blocks must be aligned to physical flash sectors. That means that a block shall start at a physical flash sector and shall comprise an integer number of flash sectors. This requirement makes sure that each logical block can be erased and reprogrammed

individually without affecting other blocks. For the same reason, logical blocks must not overlap, and they must not share one common physical flash sector.
(R4204) Logical blocks must not be nested.
(R4205) Logical blocks are handled by the so called “logical block table”. The logical block table is a list of all logical blocks of an ECU that contains an entry for each logical block. One entry of the logical block table consists of the block number, the start address and the length information of the concerned block.
(R4206) In one ECU at least one logical block must be available.
(R4207) Each block is identified by an individual block number. This block number is used to get the address- and length information from the logical block table and to access the administrative data of the block. Block numbering starts at “0”.
(R4208) For each logical block, a set of administrative data, also called meta data, shall be maintained by the bootloader software. This data set comprises the individual fingerprint, the flash driver version, the software identification, the CRC-Total and validation information of the block.

4.2.4 Configuration Examples

To illustrate the usage of logical blocks, two configuration examples with logical blocks are discussed here.
Configuration 1 of Figure 4-2 shows the ECU flash memory with the bootloader code in the protected memory area and one logical block that contains all application code and data. The complete logical block 0 can only be erased and reprogrammed at once
Configuration 2 of Figure 4-2 shows the ECU flash memory with the bootloader code in the protected memory as in configuration 1, but the reprogrammable area consists of several logical blocks. In this example, block 1 contains pure application data so that this block can be reprogrammed individually and without changing the contents of other blocks if data in this block has changed.
Administrative effort is kept to a minimum when a logical block configuration like configuration 1 is applied, because one logical block requires just one set of identification data. When several logical blocks are used, an individual set of identification data shall be administrated by the flash reprogramming process.

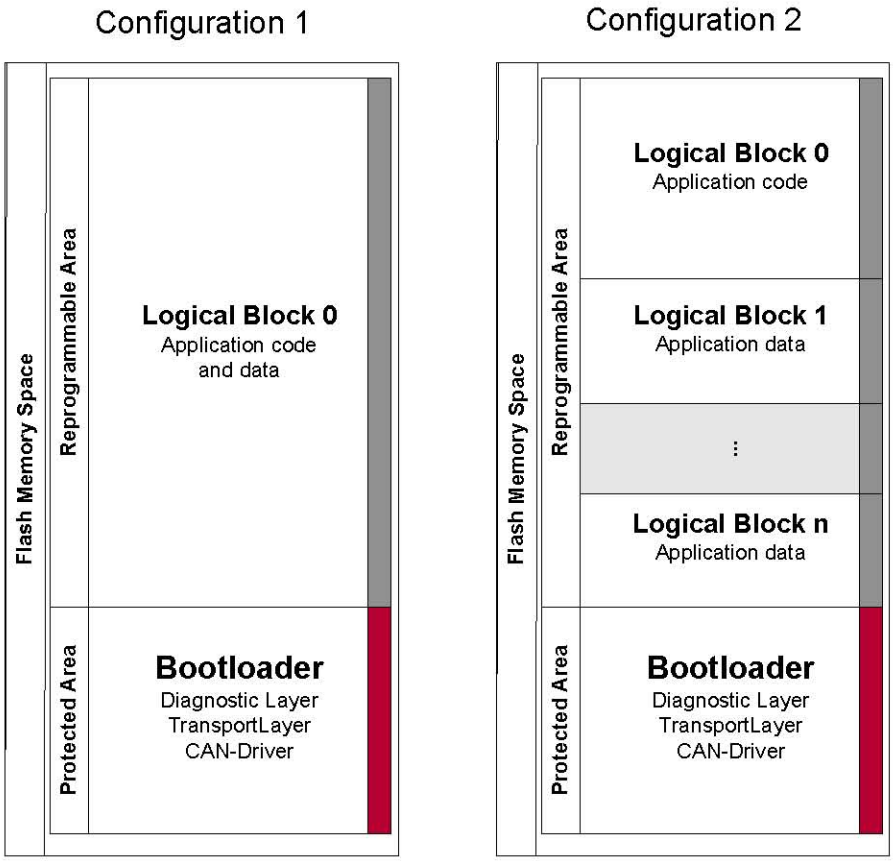

Figure 4-2 Configuration of logical blocks

4.2.5 Segmentation

The contents of logical blocks may consist of contiguous and non-contiguous data. The code of a logical block can be segmented e.g. because the linker generates different segments for the application code-, text-, const- and may be several other segments.
(R4210) A segment must reside inside of the address range of the current logical block.
(R4211) A segment must be aligned to integer multiples of flash pages according to the requirements of the underlying flash device.
(R4212) Segments are handled by the so called “Header information”, which is listed in the “segment table”. The segment table is a list of all segments of a logical block that contains an entry for each segment.

About the detailed information of “segment table”, please see the table 3-2.

The “segment table” information is listed before the first segment of the logical block, but it is not the part of the first segment of the logical block.
One segment is downloaded by a sequence of the diagnostic service requests RequestDownload – TransferData –RequestTransferExit (see section 6.2.2.6). This is repeated for all segments of the logical block.

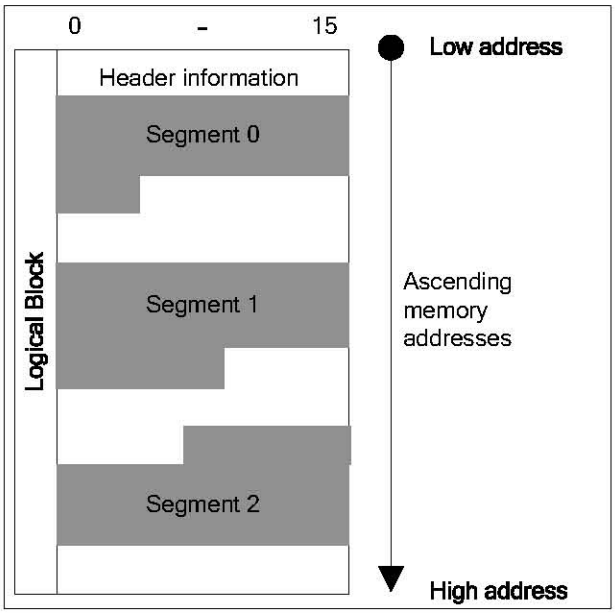

Figure 4-3 Segmentation of logical block data

4.3 ECU Startup Sequence

(R4300) After PowerOn/Reset, boot code is executed first.
(R4301) The bootloader performs some basic initializations and then checks if an external reprogramming request flag is set. In this case, bootloader code is executed furthermore, even if a valid application is available.
(R4302) If no reprogramming request is present, the status of the application is checked. If the application is valid, the bootloader starts the application. If the application is not valid, bootloader code is executed.

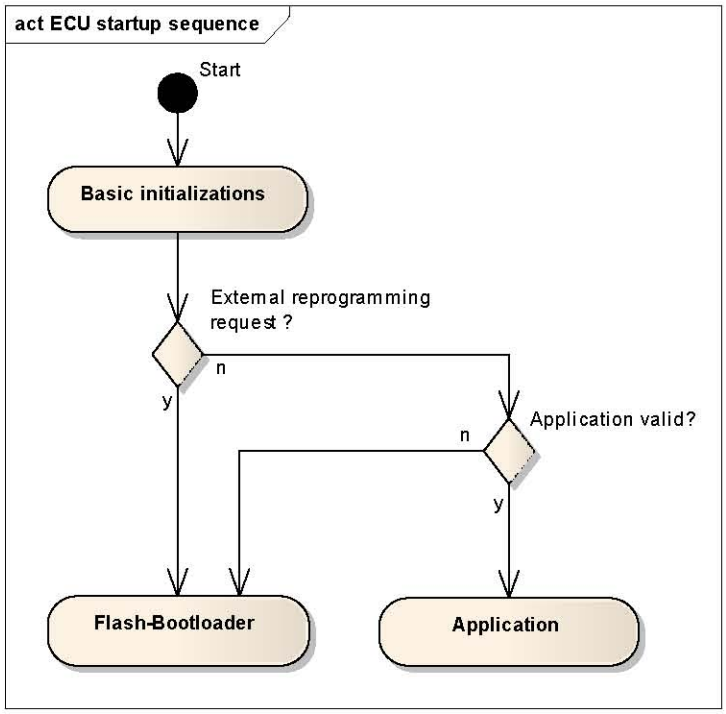

Figure 4-4 ECU startup sequence

4.3.1 Stay-In-Boot Mode

The Stay-In-Boot mode is an extension of the ECU startup sequence and avoids that an ECU is not reprogrammable in case that the application is not able to start the bootloader. Stay-In-Boot provides the ability to force the ECU into the bootloader mode after PowerOn/Reset even if the operative application software is available. This might be necessary if the application that is currently programmed on the system has got an issue that leads to a reset. In this case, the FBL starts the application again, so that an endless loop might be the consequence.
The example that is depicted in Figure 4-5 shows the endless loop that occurs when a watchdog reset is issued subsequently after the application is started by the bootloader.
The Stay-In-Boot mode is based on a wait time after PowerOn/Reset. If a particular CAN message is received during this time, the ECU stays in the bootloader and does not start the application. Otherwise, if the timer expires without receiving the CAN message, the application is started if it is valid.

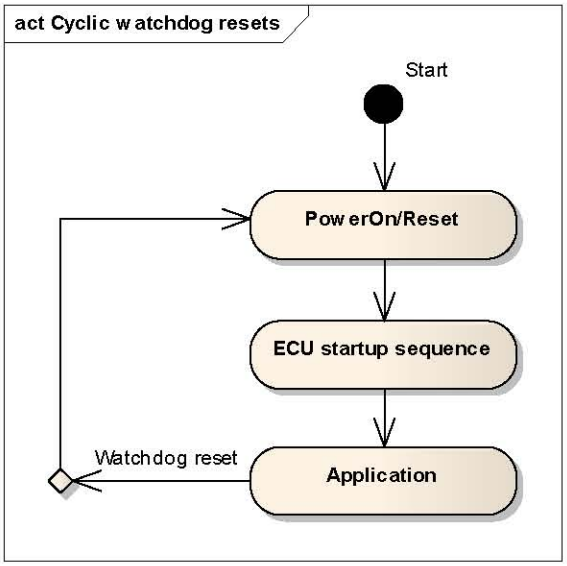

Figure 4-5 Endless loop by cyclic watchdog resets

(R4310) It is recommended that the ECU supports the Stay-In-Boot mode during development time.
(R4311) For security reasons, the Stay-In-Boot mode must be disabled for production.
(R4312) If no external reprogramming request is pending and if the application is valid, the Stay-In-Boot wait timer shall be activated.
(R4313) While the wait timer has not expired, the bootloader shall wait for the reception of the particular Stay-In-Boot CAN message. In case that this CAN message is received while the timer has not expired, the application is not started and the ECU stays in the bootloader.
(R4314) After the wait timer has expired and no Stay-In-Boot CAN message has been received meanwhile, the application shall be started.
(R4315) When the bootloader is started by Stay-In-Boot, a session timeout shall make sure, that the application is started if no communication by the tester is maintained.
(R4316) The Stay-In-Boot wait time is 20ms.
(R4317) The Stay-In-Boot CAN message identifier is the diagnostic identifier of the ECU.
(R4318) The Stay-In-Boot CAN message data content is:
DLC = 8
Data = 04 31 01 F5 18 xx xx xx
(R4319) The Stay-In-Boot CAN message is the RoutineControl – startRoutine with routine identifier $F5 $18 that shall be accepted in all sessions.
(R4320) When the Stay-In-Boot CAN message is accepted, a positive response shall be sent.

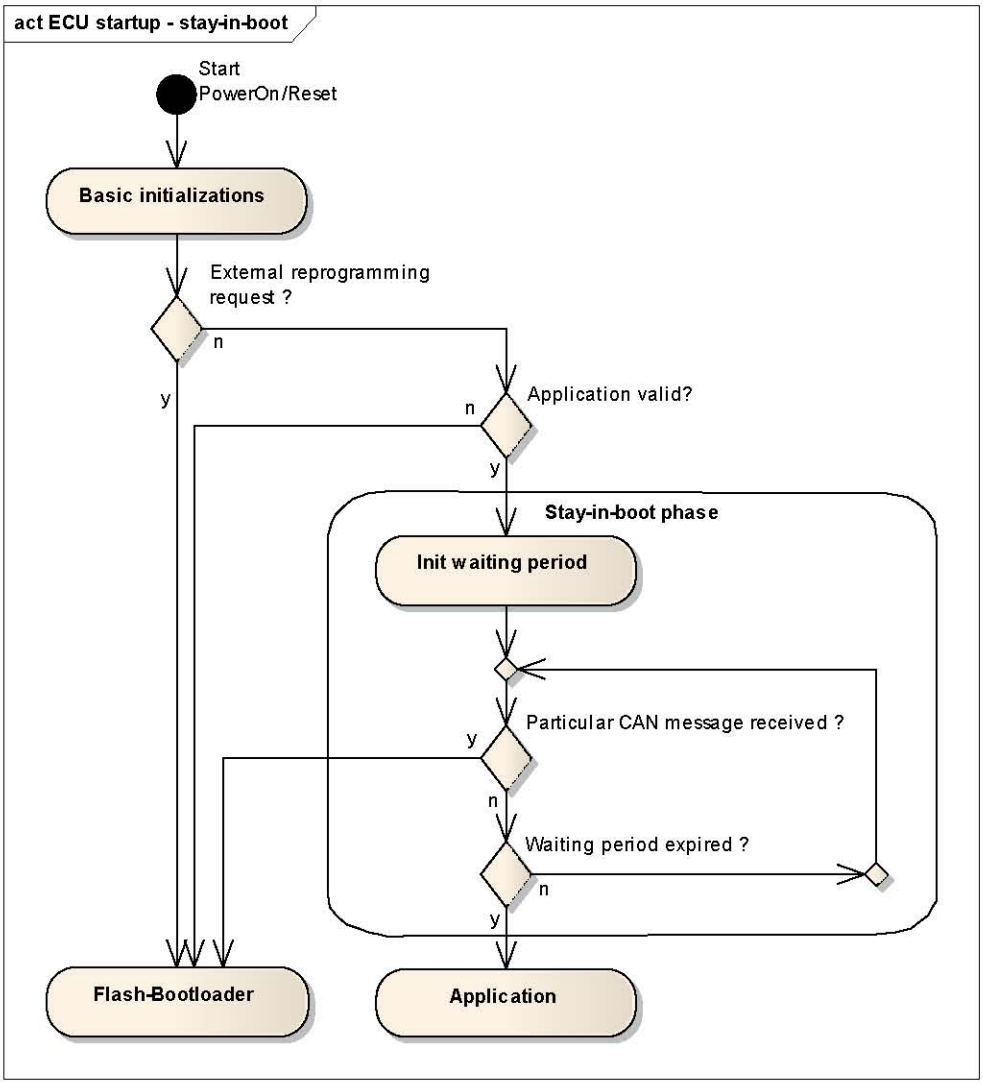

Figure 4-6 Stay-in-boot mode

4.4 Flash Driver

The flash driver is a hardware-dependent module that provides the functionality of erasing and programming the ECU flash memory.

(R4400) The flash memory contents must be secured against unintended erasure and overwriting. Therefore, a software interlock mechanism is implemented in the bootloader code that stores critical code outside of the ECU memory. The complete flash driver code or critical parts are not stored in the ECU flash memory but are downloaded into an ECU RAM buffer during the download procedure.

(R4401) After the download is completed the flash driver code shall be explicitly removed from the RAM buffer before the ECU returns to normal operation mode.

Downloading the flash driver into a RAM buffer when it is needed is an additional advantage, because flash memory resources can be saved this way. Additionally, on most microcontroller platforms, flash memory cannot be erased or programmed by code stored in flash memory or at least in the same flash bank.

The flash driver shall provide a special API to be invoked by the bootloader. At least the following four routines are required:
(R4402) Initialization: The initialization routine is called by the bootloader after the flash driver has been downloaded to the ECU to perform hardware specific initializations for flash programming.
(R4403) De-Initialization: After the download is completed, the bootloader calls the de-initialization to perform hardware-specific operations to finish flash programming.
(R4404) Erase: The erase routine is called by the bootloader to erase the requested flash area.
(R4405) Write: All download data is programmed by the bootloader using the write routine of the flash driver.
Supplier should send flash driver files to GAC in Bin file format, incuding header information. which is specified in table 4-4.

Offset	Length (byte)	Name	Description
$0	1	NOAR	Number Of Segments This field is used to accommodate programming of flash driver. The number in this field represents the number of different address regions addressed when flash driver is downloaded.
$1	4	AR1	Address of First Segment
$5	4	LR1	Length of First Segment
$9	4	AR2	Address of Second Segment
$13	4	LR2	Length of Second Segment
$25	4	AR3	Address of Third Segment
$29	4	LR3	Length of Third Segment
…	…	…	…

Table 4-4 Flash driver header information

4.5 Watchdog Support

(R4500) On most ECUs at least one watchdog is to be served in the bootloader and during flash programming. Therefore, the bootloader shall provide a watchdog support to serve either normal or windowed watchdog devices. The watchdog device can be either an internal one on the microcontroller or an external one.
Some watchdog devices permit to switch between a “short” and a “long” operating mode. Sometimes these modes are also called “fast” and “slow”.
(R4501) The bootloader shall support these operation modes by providing ECU-specific functions so that the required operations can be performed.

The bootloader shall provide the following routines for watchdog support. These routines are ECU-specific so that they can be adapted by the supplier:
(R4502) Initialization: To initialize the watchdog, the initialization function is called by the bootloader during the initialization phase.
(R4503) Switch the Watchdog to “Short”-period operation mode: When the bootloader is initialized after reset or after the ECU application activated the bootloader this function is called.
(R4504) Switch the watchdog to “Long”-period operation mode: This function is called before the bootloader causes a reset.
(R4505) Watchdog service function: The watchdog service function is called by the bootloader with the preconfigured trigger cycle time to serve the watchdog. This function consists of ECU specific code so that a hardware-specific algorithm for serving the watchdog can be implemented. Since the watchdog service function is also called by the flash driver, this function must consist of position-independent code so that this function can be copied to a RAM buffer. It is not allowed to implement function calls within this function.
(R4506) The bootloader shall support a watchdog trigger cycle of 1ms (millisecond) or larger.

4.6 Security Module

(R4600) The bootloader shall provide an interface for security functionality to secure the flash download process against

- Transmission and programming errors
- Unauthorized download attempts
- Download attempts of software that does not originate from a legitimate source
- Unauthorized reading of the ECU application code and/or -data during the deployment process of the ECU application to protect the know-how of supplier and OEM.

Depending on the required security level of an ECU, there are the following security classes to achieve the required security:

Security Class	Description
None / DDD	Provides protection against unauthorized download attempts by the SecurityAccess service and secures integrity of download data by a CRC32 algorithm.
C	In addition to the security class none/DDD, the security class C provides authenticity verification by a HMAC (Hash Message Authentication Code) algorithm.
CCC	RSA-encrypted signature
AAA	Encryption of the software

Table 4-2 Security classes

The security classes C, CCC and AAA are optional.
To cover the required security functionality, the security module of the bootloader consists of the following components:

- Seed-key functionality for SecurityAccess procedure
- CRC32 routine
- Signature verification functionality (optional)
- Decryption functionality (optional)

4.6.1 Security Access

(R4610) Flash reprogramming shall be secured against unauthorized download attempts. Therefore, the security access procedure must be passed successfully before further reprogramming steps are unlocked.
In the programming session, the seed level $11 and the key level $12 is used. For further details, please refer to [1].

4.6.2 Integrity Verification

(R4620) Every logical block is secured with a CRC32 value. After a download, the bootloader must ensure that all data bytes,not including "segment table" bytes of the current block have been transferred and written correctly. Therefore, a CRC32 algorithm as verification routine is applied. The verification routine is called by the bootloader when the RoutineControl service request with checkRoutine routine identifier is received and calculates the CRC32 of the downloaded data bytes, not including "segment table" bytes. The resulting value is compared with the reference check value that is transmitted to the ECU in the service request message. The verification routine is applied for the flash driver and for the application download.
The integrity verification shall be implemented in compliance with the Ethernet IEEE 802.3 standard specification of the CRC32.
(R4621) For the CRC32 computation, the following generator polynomial is used:
$G(x) = x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$
(R4622) The initial value is 0xFFFFFFFF.
(R4623) All bits of the result have to be inverted (XORed with 0xFFFFFFFF).
(R4624) The result value is bit-reversed.
When the CRC32 algorithm is applied to the ASCII string "123456789" (binary coded $31, $32, $33, $34, $35, $36, $37, $38, $39) the hex value CBF43926 is obtained.

4.6.3 Signature Verification

(R4630) To make sure that no unauthorized software is downloaded to the ECU, a signature based on a cryptographic hash algorithm is calculated for the downloadable data. The signature verification in the bootloader is activated with the RoutineControl service request with checkRoutine routine identifier. As with the CRC32 verification, the signature value is

transmitted to the ECU in the service request message and is compared with the value computed by the ECU.
The usage of the signature verification procedure is optional and depends on the security class of the ECU.

4.6.4 Decryption- Decompression Interface

(R4640) The bootloader provides interfaces that enable the download of encrypted and/or compressed data. Encrypted download data may be necessary when confidentiality of the software is required. Compression of download data may be helpful in order to reduce download time.

4.7 Sleep Mode

The sleep mode in the bootloader is needed to reduce power consumption when the ECU is idle and no diagnostic messages are received and no valid application could be started.
(R4670) The bootloader enters sleep mode after an internal sleep counter has expired. The initial sleep counter value is 300 seconds.
(R4671) The internal sleep counter is started during the bootloader initialization procedure.
(R4672) With each diagnostic message received by the ECU the sleep counter is reset.
(R4673) When the sleep counter expires, the bootloader shall enter the sleep mode. The sleep mode shall be implemented as required by the specific hardware in order to wake up correctly on any activity, e.g. CAN communication or ignition-on.

5 Network Layer Parameters

5.1 CAN Communication Parameters

(R5100) The addressing scheme and the CAN Identifier assignment are available in [4].
(R5101) DLC of the download CAN-message is fixed to 8 bytes.

5.2 Transport Layer Parameters

(R5200) The transport layer implementation for flash download shall be based on ISO 15765-2 [3].
(R5201) The transport layer timing parameters (N_As, N_Br, etc.) shall be set as specified in [1].
(R5202) Blockszie and STmin shall be set as specified in [1].

6 Diagnostic Communication

Flash reprogramming is controlled by a subset of the diagnostic communication protocol. This chapter specifies the requirements for diagnostic communication like session management, the reprogramming sequence and the relevant diagnostic services.

6.1 Diagnostic Session Management

In connection with the flash reprogramming, there are three different diagnostic sessions used, the default session, the extended diagnostic session and the programming session.

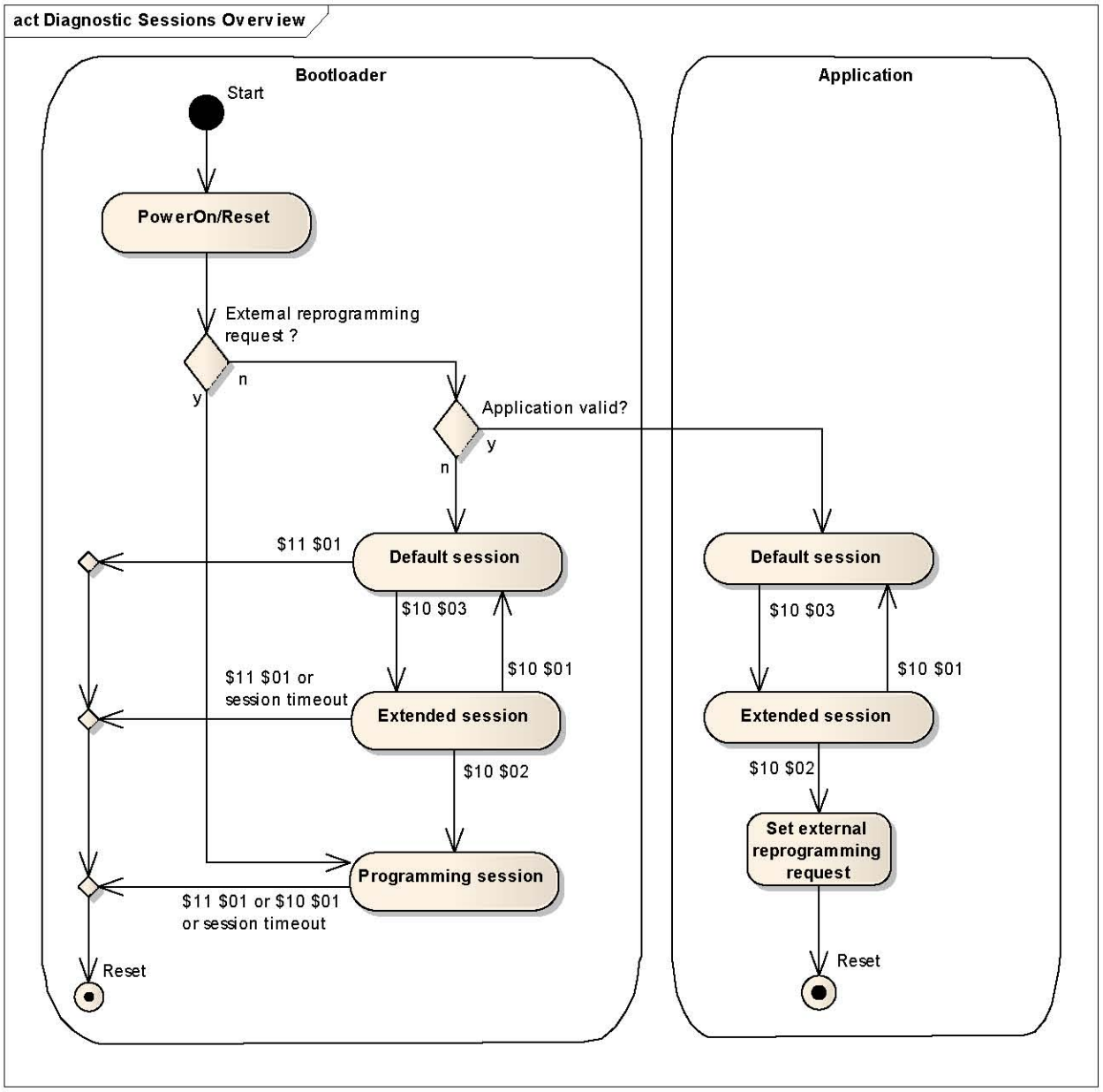

Figure 6-1 Diagnostic sessions overview

After a PowerOn/Reset occurred, the ECU is started as described in section 4.3.

(R6100) If no external reprogramming request is present, the bootloader determines if a valid application is available. If a valid application is detected, the application is started in default session.
(R6101) If the application is invalid, the ECU continues executing bootloader code in the default session and waits for requests to switch to extended session and to programming session for ECU reprogramming.
(R6102) The extended session of the bootloader can be left by a default session request so that the default session of the bootloader is entered.
(R6103) If the ECU is executing its operative application software in the extended diagnostic session, a DiagnosticSessionControl service request with programmingSession sub parameter leads to the activation of the bootloader. Therefore, the application shall set the external reprogramming request flag and shall issue a Reset. Upon the Reset, the ECU is starting-up as specified in Figure 6-1. When the bootloader is started by the application with $10 $02, the programming session of the bootloader is active.
(R6104) The programming session will be left by a session timeout, by an ECUReset or a defaultSession service request.
(R6105) The session transition request from programming session to the extended session within the Bootloader shall not be supported.

6.2 Flash Reprogramming Sequence

The flash reprogramming sequence is divided into three steps, the pre-programming step, the server programming step and the post-programming step. These three steps are described in detail in the following sections. In the sequences depicted in Figure 6-2, Figure 6-3, and Figure 6-4, optional steps are shown on the right side, similar to the graphics in the ISO document.

6.2.1 Pre-Programming Step

(R6200) The pre-programming step is used to prepare the network for the programming procedure.

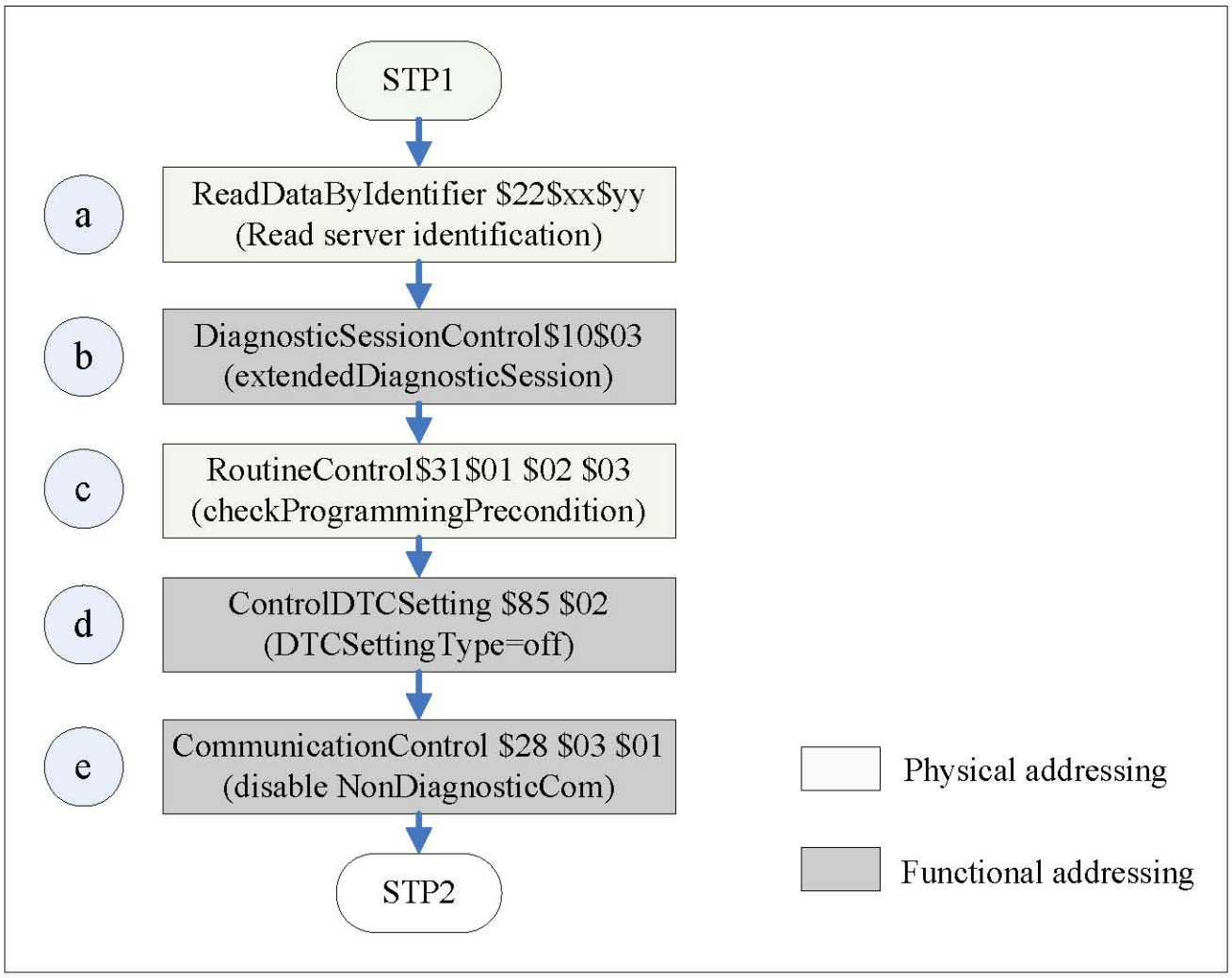

Figure 6-2 Pre-programming step

6.2.1.1 (a) ReadDataByIdentifier $22 $xx $yy

The ReadDataByIdentifier service is used to acquire identification information or other special information from the ECU that is be reprogrammed. ECU identification information or other special information is applied by the flash process infrastructure to determine the suitable software update that shall be programmed and to document the download event.

6.2.1.2 (b) DiagnosticSessionControl $10 $03

Before an ECU can be reprogrammed, the setting of DTCs and the normal communication in the network shall be disabled. Therefore, a non-default diagnostic session shall be started on all ECUs connected to the network. Therefore, this request uses functional addressing. With this service request the external reprogramming tool shall start transmitting TesterPresent messages in order to keep all ECU of the network in the extended diagnostic session.

6.2.1.3 (c) RoutineControl $31 $01 $02 $03

The check of programming preconditions is required to make sure that the system is in a safe state so that the reprogramming procedure can be carried out.

6.2.1.4 (d) ControlDTCSetting $85 $02

With a ControlDTCSetting service request with DTCSettingType set to “off” to disable detection and storage of DTCs by an ECU during the reprogramming procedure. This request uses functional addressing to disable fault code setting on all ECUs of the network.

6.2.1.5 (e) CommunicationControl $28 $03 $01

With a CommunicationControl service request, all ECUs connected to the network disable transmission of normal application-related communication, non-diagnostic messages, not including network management related communication messages. Having normal message transmission disabled, an ECU does not receive and processes incoming messages and does not send messages to other ECU. By disabling normal message transmission, the full bandwidth of the bus is available for the download and the download is not disturbed by non-diagnostic messages.

6.2.2 Server Programming Step

(R6210) The server programming step is used to program one or several logical blocks. All service requests of this step use physical addressing, so that it is possible to program multiple nodes in parallel. Several logical blocks can be programmed without terminating the server programming step. The download of one logical block is initiated by the WriteDataByIdentifier service request and is terminated by the successful verification of the download by the RoutineControl with the checkRoutine routine identifier.

(R6211) If an error occurs during the server programming step, the following conditions shall be applied to restart the sequence:

If the error occurred in request a) or b), the download shall be restarted at the current service. If the error occurred with the requests c) – g), the restart shall be done at c). If the second attempt fails again, a physically addressed EcuReset – hardReset ($11 $01) shall be sent to the ECU and the full sequence shall be performed again.

If request h) fails, an incompatible module version has been downloaded. In this case, the incompatible module has to be identified and downloaded again with the full sequence.

If a timeout occurs while processing the transferData service ($36), the block sequence counter shall be used for improved error handling as specified in [2].

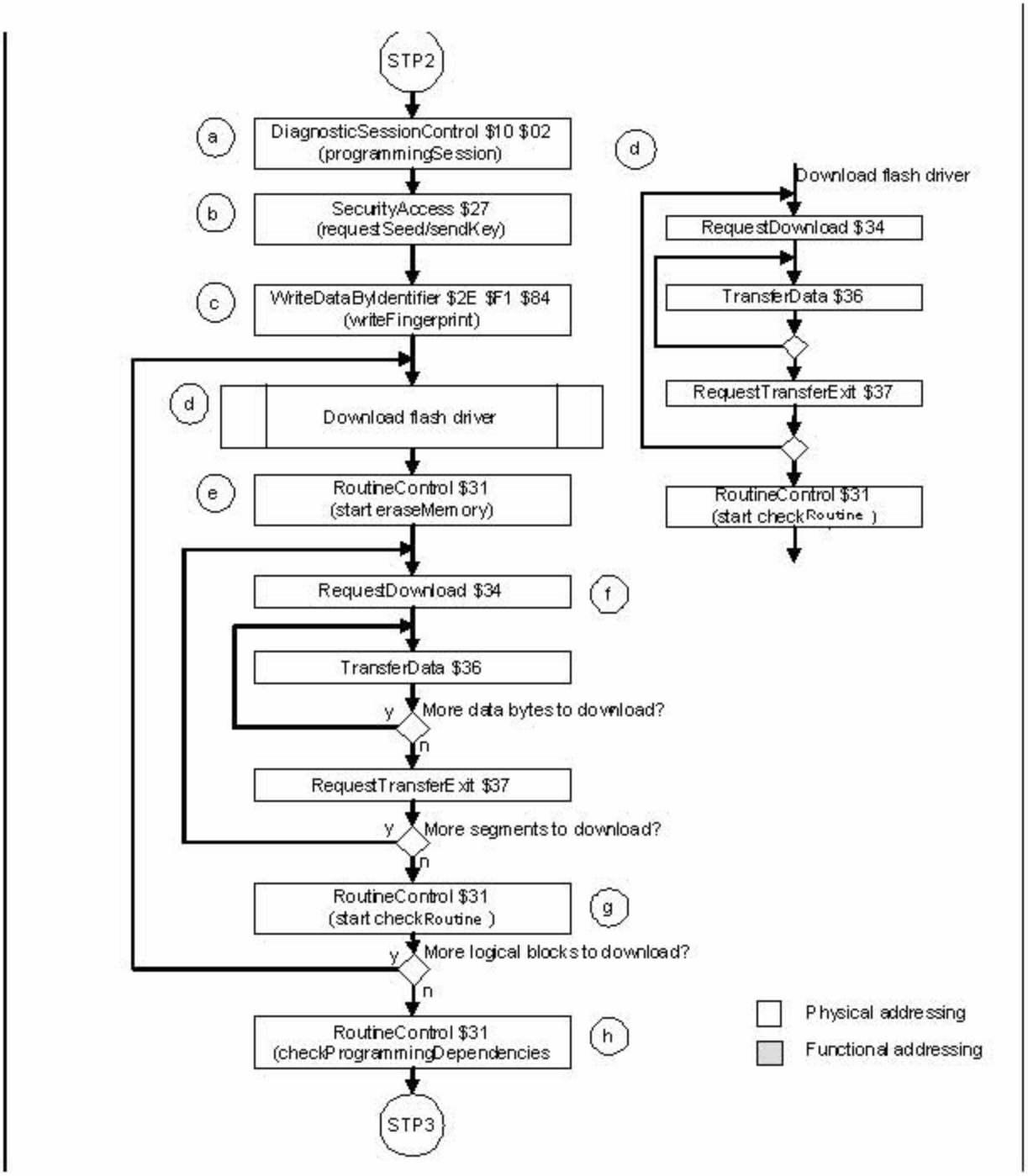

Figure 6-3 Programming step

6.2.2.1 (a) DiagnosticSessionControl $10 $02

The download procedure is started in the ECU with a DiagnosticSessionControl service request with a sub function parameter set to programmingSession. In an ECU, that is executing its operative application software, this service request leads to a transition from application software to bootloader software. While the reprogramming session is active, reprogramming specific service requests are supported. Additionally, ECU should have

executed step1 before transition from application software to bootloader software, otherwise, ECU will send negative response for request $10 $02 with negative code $22(ConditionsNotCorrect).

6.2.2.2 (b) SecurityAccess $27 $11/$12

Security access shall be granted to the tester by the ECU before flash reprogramming. To unlock an ECU, the tester must request the seed from the ECU first. Then, both the tester and the ECU calculate the key and the tester transmits it to the ECU. The ECU compares its key value with the one received from the tester and if the values match, security access will be granted to the tester.

6.2.2.3 (c) WriteDataByIdentifier $2E $F1 $84

Prior to any flash access, the fingerprint of the tester shall be stored on the ECU's non-volatile memory. The fingerprint is transferred to the ECU by a WriteDataByIdentifier service request.
The fingerprint is a mandatory information structure for the reprogramming process that identifies a certain download or a download attempt. A fingerprint is necessary to be able to track a download attempt or a complete download.

6.2.2.4 (d) Download of Flash Driver (Software Interlock) $34, $36, $37, $31

This is a mandatory step to download the flash driver into the designated RAM buffer of the ECU. Flash tool shall acquire memory address and length from flash driver"header information", which is listed in table 4-4.
This step consists of several service requests, RequestDownload, TransferData and RequestTransferExit. After all bytes are transferred, a verification procedure is started with a RoutineControl service request with sub parameter $01 for startRoutine and routine id $02 $02 - checkRoutine to ensure that all bytes are copied correctly.

6.2.2.5 (e) RoutineControl – Start eraseMemory $31 $01 $FF $00

The RoutineControl service request with parameter $01 $FF $00 erases the requested logical block. Before the erase routine of the flash driver is called, the validity status of the logical block in question must be set to invalid. This prevents accidental execution of the application if the flash process does not terminate successfully. Additionally, the fingerprint that has been received with a previous WriteDataByIdentifier service request shall be stored in non-volatile memory before the logical block is erased.

6.2.2.6 (f) Download Process $34, $36, $37

All segments of a logical block are downloaded to the ECU by a sequence of the services RequestDownload, TransferData and RequestTransferExit. The download of one segment that consists of a contiguous set of data bytes is started with a RequestDownload service

request. The RequestDownload service informs the bootloader about the start address in memory and the length of the segment. After RequestDownload, all data bytes of the segment are transferred by one or more subsequent TransferData requests. After all segment bytes are transferred, the download of a segment is terminated with a RequestTransferExit request. This sequence can be repeated for several times depending on the number of segments in a logical block.

6.2.2.7 (g) RoutineControl – Start checkRoutine $31 $01 $02 $02

After all bytes of a logical block are transferred to the ECU, the verification routine is started to ensure integrity (and optionally authenticity) of the downloaded data.
If there is another logical block to be downloaded, the server programming sequence can be continued with service request (d) - Download flash driver. After the download of the last logical block, the sequence is continued with the post-programming step.

6.2.2.8 (h) RoutineControl – Start checkProgrammingDependencies $31 $01 $FF $01

To make sure that the reprogrammed logical blocks are consistent and compatible, i.e. that all blocks e.g. use compatible interfaces and formats, this service request is sent to the ECU after the download of the last logical block. With this service, checks can be implemented to make sure, that e.g. the application data is compatible to the application software or that the interfaces of two software modules match. There is no standard method to implement this check, it is dependent on the needs of the ECU application. .
Only if this check is successful, the application can be started.
If only one logical block is used on an ECU, the routineStatusRecord of the positive response is always $00 – correctResult (see also 6.3.3.7.4).

6.2.3 Post-Programming Step

(R6220) The post programming step is performed to conclude the programming event after the programming step of a reprogrammed ECU is finished.

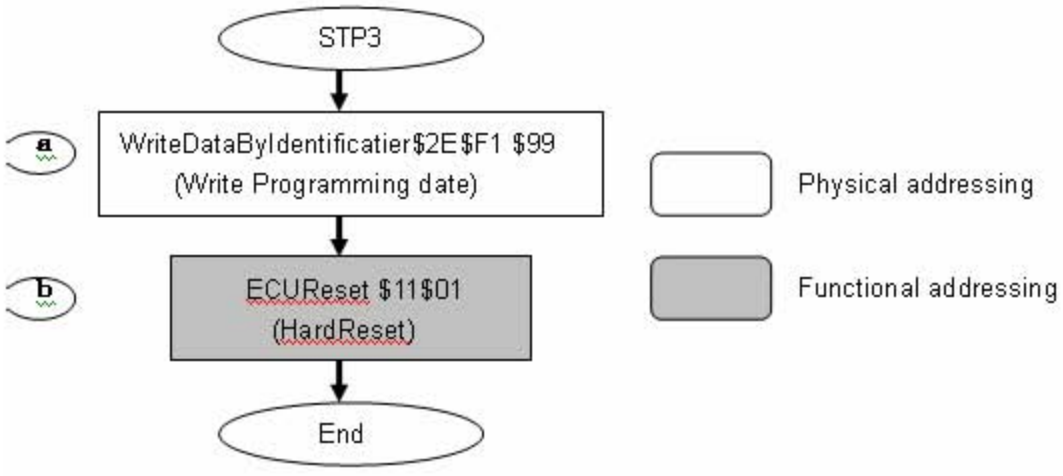

Figure 6-4 Post-programming step

6.2.3.1 (a) WriteDataByIdentifier $2E $F1 $99

The WriteDataByIdentifier service is used to write programming date to the ECU after successful download.
It shall be stored on the ECU's non-volatile memory. This identification can be able to track a successful download.

6.2.3.2 (b) EcuReset $11 $01 (functional addressing)

The flash reprogramming process is terminated by an ECUReset service request in order to return to normal ECU operation.
The flash driver code must be removed completely from its RAM buffer to avoid accidental activation of the code that might end up in unintended erase or program operations.

6.3 Diagnostic Services Overview

6.3.1 Diagnostic Services in Bootloader Software

(R6300) This chapter gives an overview of the UDS diagnostic services subset that is required in the boot software. The following table shows the required service Ids and the supported sub function parameters. Additionally, the table shows if a certain service request is addressed physically, functionally or both. The last three columns indicate if a service is supported in the default-, programming or extended diagnostic session.

x: Service request supported for phys. / funct. request; request supported in session XY

-: Service request not supported for phys. / funct. request; request not supported in session XY

Diagnostic Service Description	Hex value	Phys. Req.	Funct. req	Supported in session		
Sub function Parameter				$01	$02	$03
DiagnosticSessionControl	**10**					
Default Session	10 01	x	x	x	x	x
Programming Session	10 02	x	-	-	x	x
Extended Session	10 03	x	x	x	-	x
ECUReset	**11**					
Power On Reset	11 01	x	x	x	x	x
ReadDataByIdentifier	**22**					
Data Identifier	22 xx yy	x	-	x	x	x
SecurityAccess	**27**					
Request Seed (for reprogramming)	27 11	x	-	-	x	-

Send Key (for reprogramming)	27 12	x	-	-	x	-
CommunicationControl	**28**					
disableRxAndTx	28 03 01	x	x	-	x	x
enableRxAndTx	28 00 01	x	x	-	x	x
WriteDataByIdentifier	**2E**					
applicationSoftwareFingerprintDataIdentifier	2E F1 84	x	-	-	x	-
Programming date	2E F1 99	x	-	-	x	-
RoutineControl	**31**					
CheckProgrammingPreconditions	31 01 02 03	x	x	-	-	x
Start Erase Memory	31 01 FF 00	x	-	-	x	-
Start Check Routine	31 01 02 02	x	-	-	x	-
CheckProgrammingDependencies	31 01 FF 01	x	-	-	x	-
stayInBoot	31 01 F5 18	x	-	x	x	x
RequestDownload	**34**					
No sub function parameter	-	x	-	-	x	-
TransferData	**36**					
Block Sequence Counter	36 xx	x	-	-	x	-
RequestTransferExit	**37**					
No sub function parameter	-	x	-	-	x	-
Tester Present	**3E**					
ZeroSubFunction	00	x	x	x	x	x
ControlDTCSetting	**85**					
DTCSettingType = on	85 01	x	x	-	-	x
DTCSettingType = off	85 02	x	x	-	-	x

Table 6-1 Diagnostic services in boot software

6.3.1.1 Diagnostic Services with Sub Function Parameter

(R6301) The following table shows the diagnostic services with sub function parameter that are supported in the bootloader. These services shall support the

suppressPosRspMsgIndication-Bit as specified in [2]. The ECU shall strictly keep to the status of the flag in the request message. It is in the responsibility of the tester to use the flag in connection with those services that do not require data from the ECU in the positive response.

Diagnostic Service Description	SID
	Hex value
DiagnosticSessionControl	10
ECUReset	11
SecurityAccess	27
CommunicationControl	28
RoutineControl	31
Tester Present	3E
ControlDTCSetting	85

Table 6-2 Diagnostic services with sub function parameters

6.3.2 Diagnostic Services in Application Software

(R6310) There are several diagnostic services that are necessary in the application software to support flash reprogramming. These services are shown in Table 6-3.

Diagnostic Service Description	Hex value	Phys. Req.	Funct. req	Supported in session	
Sub function Parameter				$01	$03
DiagnosticSessionControl	**10**				
Default Session	10 01	x	x	x	x
Programming Session	10 02	x	-	-	x
Extended Session	10 03	x	x	x	x
ECUReset	**11**				
Power On Reset	11 01	x	x	x	x
ReadDataByIdentifier	**22**				
Data Identifier	22 xx yy	x	-	x	x
CommunicationControl	**28**				
disableRxAndTx	28 03 01	x	x	-	x
enableRxAndTx	28 00 01	x	x	-	x
ControlDTCSetting	**85**				

DTCSettingType = on	85 01	x	x	-	x
DTCSettingType = off	85 02	x	x	-	x

Table 6-3 Diagnostic services in application software

6.3.3 Diagnostic Services and Formats Supported in the Bootloader

This chapter specifies the diagnostic sub function parameters and message formats that are supported in the bootloader. This is a subset of the diagnostic communication services specified in [1].

6.3.3.1 DiagnosticSessionControl $10

(R6320) The service format is applied as specified in [1].

6.3.3.2 ECUReset $11

(R6321) The sub function parameter “Reset Mode” supported in the bootloader is “PowerOnReset” ($01). Other values are not supported.

6.3.3.3 ReadDataByIdentifier $22

(R6322) The service format is applied as specified in [1]. In the bootloader, only a sub set of data identifiers is supported.
For response format specification, please refer to [1].

6.3.3.4 SecurityAccess $27

(R6324) The security access service in the bootloader software uses security level code $11 for the request of a reprogramming seed and $12 to send the reprogramming key. Other security level codes are not supported in the bootloader. For further information, please refer to [1].
(R6325) The following services shall only be accessible after a successful security access unlock procedure: $2E, $31 – checkRoutine, $31 –eraseMemory, $31 – checkReprogrammingDependencis, $34, $36, $37.

Data byte	Parameter name	Hex value
#0	Security Access Request Service Id	27
#1	Security Level Request Reprogramming Seed Send Reprogramming Key	 11 12
#2-#5	Security Access Data Record Access Mode = requestSeed : no data record Access Mode = sendKey: Data record contains n key bytes	00-FF

Table 6-4 Security Access request message definition

6.3.3.4.1 SecurityAccess – "requestSeed"

(R6326) Service request message definition for subfunction $11 – "requestSeed".

Data byte	Parameter name	Hex value
#0	Security Access Request Service Id	27
#1	Security Level Request Reprogramming Seed	 11

Table 6-5 SecurityAccess – "requestSeed" service request format

(R6327) Positive response message format for for subfunction $11 – "requestSeed".

Data byte	Parameter name	Hex value
#0	Security Access Request Service Id	67
#1	Security Level Request Reprogramming Seed	 11
#2-n	Security Access Data Record securitySeed - n bytes	 00-FF

Table 6-6 SecurityAccess – "requestSeed" response format

6.3.3.4.2 SecurityAccess – "sendKey"

(R6328) Service request message definition for subfunction $12 – "sendKey"

Data byte	Parameter name	Hex value
#0	Security Access Request Service Id	27
#1	Security Level Request Reprogramming Seed	 12
#2-n	securityKey, n bytes	00-FF

Table 6-7 SecurityAccess – "sendKey" service request format

(R6329) Positive response message format for for subfunction $12 – "sendKey".

Data byte	Parameter name	Hex value
#0	Security Access Request Service Id	67
#1	Security Level Request Reprogramming Seed	 12

Table 6-8 SecurityAccess – "sendKey" response format

6.3.3.5 CommunicationControl $28

(R6330) The service format is applied as specified in [1] but shall be restricted to the parameter values in Table 6-9.

Data byte	Parameter name	Hex value
#0	CommunicationControl Request Service Id	28
#1	controlType enableRxAndTx disableRxAndTx	 00 03
#2	communicationType	01

Table 6-9 CommunicationControl service request format

6.3.3.6 WriteDataByIdentifier $2E

(R6331) The WriteDataByIdentifier service in the bootloader supports the data identifier $F1 $84 – "Fingerprint", and $F1 $99 – "programming date".

6.3.3.6.1 WriteDataByIdentifier – "Write Fingerprint"

(R6331-1) Service request message definition for data identifier $F1 $84 –"Fingerprint".

Data byte	Parameter name	Hex value
#0	WriteDataByIdentifier Request Service ID	2E
#1	Fingerprint DID (MSB)	F1
#2	Fingerprint DID (LSB)	84
#3	Tool Suplier Identification	00-FF
#4	programmingDate YY (byte 0, BCD-coded)	00-99
#5	programmingDate MM (byte 1, (BCD-coded)	00-12
#6	programmingDate DD (byte 2, BCD-coded)	00-31
#7	testerSerialNumber (byte 0)	00-FF
#8-11	testerSerialNumber (bytes 1-4)	00-FF
#12	testerSerialNumber (byte 5)	00-FF

Table 6-10-1 WriteDataByIdentifier – "Fingerprint"

(R6331-2) Service request message definition for data identifier $F1 $99 –"programming date".

Data byte	Parameter name	Hex value
#0	WriteDataByIdentifier Request Service ID	2E
#1	Programming Date DID (MSB)	F1
#2	Programming Date DID (LSB)	99

#3	programmingDate YY (byte 0, BCD-coded)	00-99
#4	programmingDate YY (byte 1, BCD-coded)	00-99
#5	programmingDate MM (byte 2, (BCD-coded)	00-12
#6	programmingDate DD (byte 3, BCD-coded)	00-31

Table 6-11 WriteDataByIdentifier – "programming date"

6.3.3.7 RoutineControl $31

(R6332) The RoutineControl service in the bootloader is supported for the routine identifiers $02 $02 – "checkRoutine", $02 $03 – "checkProgrammingPreconditions", $FF $00 – "eraseMemory" and $FF $01 – "checkProgrammingDependencies".

(R6333) The bootloader software shall only support routineControlType $01 – "startRoutine".

Data byte	Parameter name	Hex value
#0	RoutineControl Request Service Id	31
#1	routineControlType startRoutine	 01
#2 - #3	routineIdentifier checkRoutine checkProgrammingPreconditions eraseMemory checkProgrammingDependencies	 0202 0203 FF00 FF01
#4 - n	routineControlOptionRecord	00-FF

Table 6-12 RoutineControl service request format

6.3.3.7.1 RoutineControl – "checkRoutine"

(R6334) Service request message definition for routine identifier $02 $02 – "checkRoutine".

Data byte	Parameter name	Hex value
#0	RoutineControl Request Service Id	31
#1	routineControlType startRoutine	 01
#2 - #3	routineIdentifier checkRoutine	 0202
#4 -#7	CRC32 value, 4 bytes	00-FF

Table 6-13 RoutineControl – "checkRoutine" service request format

(R6335) Positive response format for routine identifier $02 $02 – "checkRoutine".

Data byte	Parameter name	Hex value
#0	RoutineControl Response Service Id	71

#1	routineControlType startRoutine	 01
#2 - #3	routineIdentifier checkRoutine	 0202
#4	routineStatusRecord correctResult incorrectResult	 00 01[a]

a When the CRC32 value is not correct,ECU will send positive response with "01".

Table 6-14 RoutineControl – "checkRoutine" response format

6.3.3.7.2 RoutineControl – "checkProgrammingPreconditions"

(R6336) Service request message definition for routine identifier $02 $03 – "checkProgrammingPreconditions"

Data byte	Parameter name	Hex value
#0	RoutineControl Request Service Id	31
#1	routineControlType startRoutine	 01
#2 - #3	routineIdentifier checkProgrammingPreconditions	 0203

Table 6-15 RoutineControl – "checkProgrammingPreconditions" service request format

(R6337) Positive response format for routine identifier $02 $03 – "checkProgrammingPreconditions".

Data byte	Parameter name	Hex value
#0	RoutineControl Response Service Id	71
#1	routineControlType startRoutine	 01
#2 - #3	routineIdentifier checkProgrammingPreconditions	 0203

Table 6-16 RoutineControl – "checkProgrammingPreconditions" response format

6.3.3.7.3 RoutineControl – "eraseMemory"

(R6338) Service request message definition for routine identifier $FF $00 – "eraseMemory".

Data byte	Parameter name	Hex value
#0	RoutineControl Request Service Id	31

#1	routineControlType startRoutine	 01
#2 - #3	routineIdentifier eraseMemory	 FF00
#4	addressAndLength FormatIdentifier lengthFormat: bit 7 – 4: number of bytes of the memorySize parameter addressFormat: bit 3- 0: number of bytes of the memoryAddress parameter Recommended fix value	 0x-4x X0-x4 44
#5 – n1	memoryAddress 1 – 4 bytes erase address	00 - FF
n2 – n3	memorySize 1 – 4 bytes erase size	00 - FF

Table 6-17 RoutineControl – "eraseMemory" service request format

(R6339) Positive response format for routine identifier $FF $00 – "eraseMemory".

Data byte	Parameter name	Hex value
#0	RoutineControl Response Service Id	71
#1	routineControlType startRoutine	 01
#2 - #3	routineIdentifier eraseMemory	 FF00
#4	routineStatusRecord correctResult incorrectResult	 00 01[b]

b ECU has executed erase operation, but unsuccessfully.

Table 6-18 RoutineControl – "eraseMemory" response format

6.3.3.7.4 RoutineControl – "checkProgrammingDependencies"

(R6340) Service request message definition for routine identifier $FF $01 – "check-ProgrammingDependencies".

Data byte	Parameter name	Hex value
#0	RoutineControl Response Service Id	31
#1	routineControlType startRoutine	 01
#2 - #3	routineIdentifier checkProgrammingDependencies	 FF01

Table 6-19 RoutineControl – "checkProgrammingDependencies" service request format

(R6341) Positive response format for routine identifier $FF $01 – "check-ProgrammingDependencies".

Data byte	Parameter name	Hex value
#0	RoutineControl Response Service Id	71
#1	routineControlType startRoutine	 01
#2 - #3	routineIdentifier checkProgrammingDependencies	 FF01
#4	routineStatusRecord correctResult incorrectResult	 00 01[c]

Table 6-20 RoutineControl – "checkProgrammingDependencies" response format

c Dependencies are not consistent.

6.3.3.7.5 RoutineControl – "stayInBoot"

(R6342) Service request message definition for routine identifier $F5 $18 – "stayInBoot".

Data byte	Parameter name	Hex value
#0	RoutineControl Response Service Id	31
#1	routineControlType startRoutine	 01
#2 - #3	routineIdentifier stayInBoot	 F518

Table 6-21 RoutineControl – "stayInBoot" service request format

(R6343) Positive response format for routine identifier $F5 $18 – "stayInBoot".

Data byte	Parameter name	Hex value
#0	RoutineControl Response Service Id	71
#1	routineControlType startRoutine	 01
#2 - #3	routineIdentifier stayInBoot	 F518

Table 6-22 RoutineControl – "stayInBoot" response format

6.3.3.8 RequestDownload $34

(R6350) The service format is applied as specified in [1].

6.3.3.9 RequestUpload $35

(R6351) For security reasons, the RequestUpload service is not supported.

6.3.3.10 TransferData $36

(R6352) The service format is applied as specified in [1]. The block sequence counter shall be treated as specified in [2].

6.3.3.11 RequestTransferExit $37

(R6353) The service format is applied as specified in [1].

6.3.3.12 TesterPresent $3E

(R6354) The service format is applied as specified in [1].

6.3.3.13 ControlDTCSetting $85

(R6355) The service format is applied as specified in [1].

6.4 Transition from Application Code to Bootloader

During normal operation, the application software is executed on the ECU. This section specifies the requirements for the invocation of the bootloader when the system shall be reprogrammed.

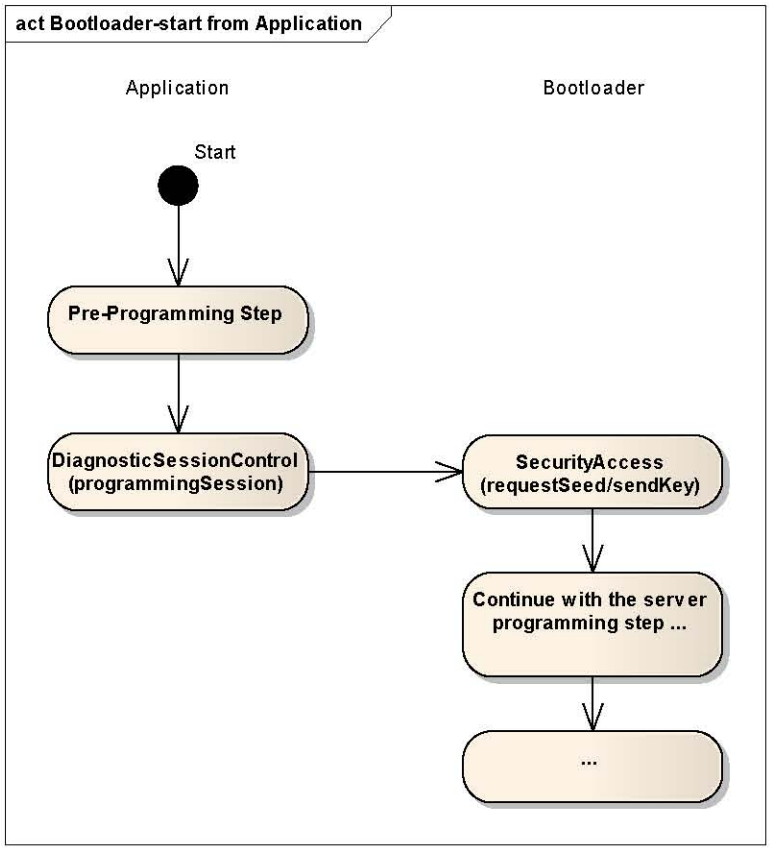

Figure 6-5 Bootloader-start from application

(R6400) When it is necessary to update the application software currently stored on the ECU, the bootloader code shall be invoked by the application. This is done by the application when the DiagnosticSessionControl service request for the programming session is received, as shown in Figure 6-5.
(R6401) The invocation of the bootloader is done by a hardware reset so that the ECU is reinitialized. This makes sure that the bootloader is not influenced by application-specific hardware settings.

6.5 Gateway Requirements

(R6500) A gateway has to route the diagnostic communication between the network to which the diagnostic tool is connected to and the target network(s). Physical diagnostic request messages which are addressed to the gateway itself needn't be routed. Functional diagnostic request messages have to routed and also have to be processed by the gateway diagnostic software component. Since the addressing scheme is normal addressing, a gateway ECU has got its own diagnostic CAN identifier pair (request and transmit message) so that the gateway application can decide if the messages have to be routed or if they are addressed to the gateway itself. If the gateway receives a StartDiagnosticSession service request for programming session transmitted using the gateway receive message identifier, the gateway enters its bootloader for reprogramming the gateway application. For that purpose, no special code is necessary in the gateway bootloader, but as a difference to the bootloader in a non-gateway ECU, the bootloader in a gateway shall route or generate TesterPresent messages on the networks connected to the gateway.

7 Reprogramming Process Related Data Requirements

The flash process requires storing a set of non-volatile data. This chapter is a proposal for how to handle this information.

The non-volatile reprogramming information block consists of two parts - the general bootloader status information and the meta data table. The general information block stores flags according to reprogramming requests, validity status of logical blocks, etc. The meta data table provides information for each logical block like fingerprint, reprogramming counter and CRC.

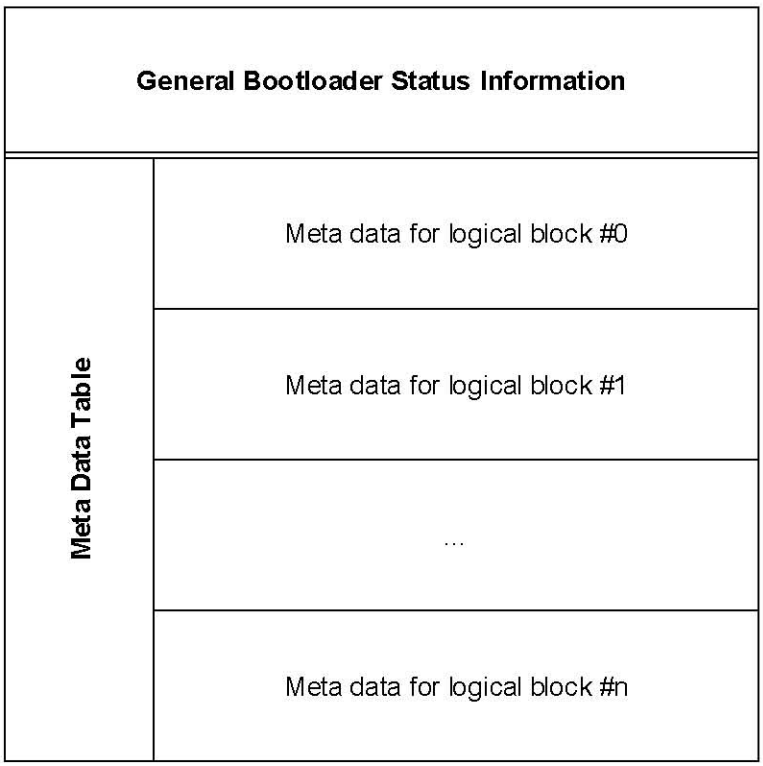

Figure 7-1 Non-volatile reprogramming information

7.1 General Non-Volatile Bootloader Status Information

The system shall store the following fields in the non-volatile memory. Each field in

Offset /bytes	Size /bytes	Field	Coding/ hex	Description
+0	1	External reprogramming request flag	00/FF B5	No external reprogramming request present The external reprogramming request flag indicates that the ECU application started the bootloader for reprogramming.
			01-B4 B6-FE	Reserved Reserved
+1	n	Validity Flags	b7-b0	Inverted validity bits of all logical blocks
			bx=0 bx=1	Logical block valid Logical block invalid
+1+n	1	Reset Response Flag		This flag specifies if a positive response must be transmitted after reset
			00/FF 01 02	Reset response not required Response for start default session ($10 $01) Response for EcuReset ($11 $01)
+2+n	1	Security Access Delay Flag	FF A7	Security access delay not active Security access delay active
			00-A6 A8-FE	Reserved Reserved
+3+n	1	Security Access Invalid Attempt Counter	FF-00	Count of failed seed-key identifications (inverted)

Table 7-1 is mandatory, but the in-memory order is an example.

Offset /bytes	Size /bytes	Field	Coding/ hex	Description
+0	1	External reprogramming request flag	00/FF B5	No external reprogramming request present The external reprogramming request flag indicates that the ECU application started the bootloader for reprogramming.

			01-B4 B6-FE	Reserved Reserved
+1	n[2]	Validity Flags	b7-b0	Inverted validity bits of all logical blocks
			bx=0 bx=1	Logical block valid Logical block invalid
+1+n	1	Reset Response Flag		This flag specifies if a positive response must be transmitted after reset
			00/FF 01 02	Reset response not required Response for start default session ($10 $01) Response for EcuReset ($11 $01)
+2+n	1	Security Access Delay Flag	FF A7	Security access delay not active Security access delay active
			00-A6 A8-FE	Reserved Reserved
+3+n	1	Security Access Invalid Attempt Counter	FF-00	Count of failed seed-key identifications (inverted)

Table 7-1 General non-volatile bootloader status information

The handling of validity information is shown in Figure 7-2 in more detail. Validity information is used by the bootloader to determine if the application can be started after PowerOn/Reset. Depending on the number of logical blocks, there are n bytes reserved to store this information. Each logical block is represented by one bit as shown in Figure 7-2.

	7	6	5	4	3	2	1	0
Validity Flags								

Figure 7-2 Validity status byte(s)

The bits b7 – b0 represent the validity of the corresponding logical block. If more than eight logical blocks are configured, additional bytes according to this schema are used.

7.2 Meta Data Table

For each logical block, the system shall store the information specified by Table 7.2.

[2] n depends on the number of logical blocks.

Offset /bytes	Size /bytes	Field	Coding /hex	Description
0	10	Fingerprint	00-FF	Flash reprogramming fingerprint
9	2	Programming Counter	0000-FFFF	Counter for successful programming sequences
11	2	Programming Attempt Counter	0000-FFFF	Counter for programming attempts
13	4	CRC	32 bit binary	CRC32 of block data
17	4	CRCstart	32 bit binary	Start address for CRC calculation
21	4	CRClength	32 bit binary	Data length for CRC calculation

Table 7 2 Non-volatile information for each logical block

The fingerprint format is specified in Table 7-3.

Offset /bytes	Size /bytes	Field	Coding /hex	Description
0	1	Tool Supllier Identification	Unsigned	
1	3	Programming Date High byte: year Mid byte: month Low byte: day	 BCD BCD BCD	Programming date
4	6	Tester Number	00-FF	Tester serial number

Table 7-3 Fingerprint format

8 Glossary and Abbreviations

8.1 Glossary

Term	Description
Client	External diagnostic tool (tester), that transmits service requests to ECUs
Fingerprint	Tester identification information that identifies a certain download attempt.
Logical Block	Reserved portion of target memory, where application data can be downloaded (like hard disk partition).
Logical Block Table	The target memory is split up in several Logical Blocks. The Logical Blocktable acts like a file system partition table. If application data is to be downloaded, the Bootloader checks, if there is a valid entry in the Logical Blocktable for the download.
Server	ECU that responds to diagnostic service request for an external diagnostic tool.
Sleep mode	Mode to reduce power consumption of the ECU in idle state.
Software Interlock	The software interlock is a protective lockout mechanism by separating critical code segments from other code in order to prevent unintended software execution e.g. after an error occurred.

8.2 Abbreviations

Abbreviation	Description
ECU	Electronic Control Unit
EEPROM	Electrically Erasable Programmable Read Only Memory
FBL	Flash Boot Loader
DTC	Diagnostic Trouble Code
CAN	Controller Area Network
CRC	Cyclic Redundancy Check
SWIL	Software Interlock
LBT	Logical Block Table
MCU	Micro Controller Unit
TP	Transport Protocol