

# Automotive BOOTLOADER

Application Software Download

0x34 : Download

0x36 : Transfer Data

0x37 : Transfer Exit

BY KAPIL  
THAKAR

# Download Request Sequence



**Programming  
Session  
Request : 10 02  
/ 10 80**



**Security  
Access to  
Unlock the ECU**



**OEM Specific  
Read Data By  
Identified**



**Erase Request**



**Programming  
Request**



**Application  
Validation**



**ECU Reset**

**Download Request :**

**0x34**

**Data Transfer :**

**0x36**

**Transfer Exit :**

**0x37**

# Format

## ► Versatile Binary Format

The Versatile Binary Format or VBF is a format used by Volvo Car Corporation and its collaborating partners [11]. The file format contains a header that contains useful information for the software loading sequence in addition to the binary data.

## ► BIN-format

In the BIN-format there is only the binary data without a structure, therefore it is not possible to easily visually inspect the data as can be done with the Motorola S format or Intel HEX. Furthermore, the format does not contain all information needed in the software loading sequence (see Chapter 2.4.2 how this is solved in the internally developed flashing tool).

## ► Intel HEX format

A binary file format that is frequently used when programming or flashing microcontrollers is the Intel HEX format. In many ways it is similar to the Motorola S-format. Containing the same kind of fields as the S-format: Fields for record type, byte count, address field, binary data and a checksum for each record.

# Format

## Motorola S-format

- ▶ The binary data contained in the SRE-format is represented according to the Motorola S-format. Motorola created the Motorola S-format for the Motorola 6800-series 8-bit microprocessors in the 1970:s. The main benefit of the S-format
- ▶ is that it provides a way to transport data in a way, which can be visually inspected. The S-format file contains several lines of S-records, each S-record contains five data fields: record type, byte count, address, binary data and checksum.
- ▶ In Figure 2-5 a typical S-format file is depicted. The S0-record is the header record for all S-format files and the S7-record is a termination record for a S-format file with S3 data records. The S3-record contains 4 address field bytes. The address field of the first S3 data record is used in the internally developed tool (see Chapter 2.4) to identify which file block is being read. There are a few different record types specified in the Motorola S-format standard with varying address field size. The byte count is denoted in hexadecimal, in this case the S3-record is 37 bytes long (25 in hex), included in this number the address field therefore there are 33 data bytes in this record (37 bytes in total – 4 address field bytes = 33 data bytes in total) [9] [10] .

S0140000286329205441534B494E472C20496E632E7272

S32500C0B400 Data bytes, 33 bytes in total... 2E42

S32500C0XXXX More S3 records..... 3A42

S32500C0XXXX More S3 records..... 0042

S32500C0XXXX More S3 records..... 0042

S7140000286329205441534B494E472C20496E632E7276

# Format

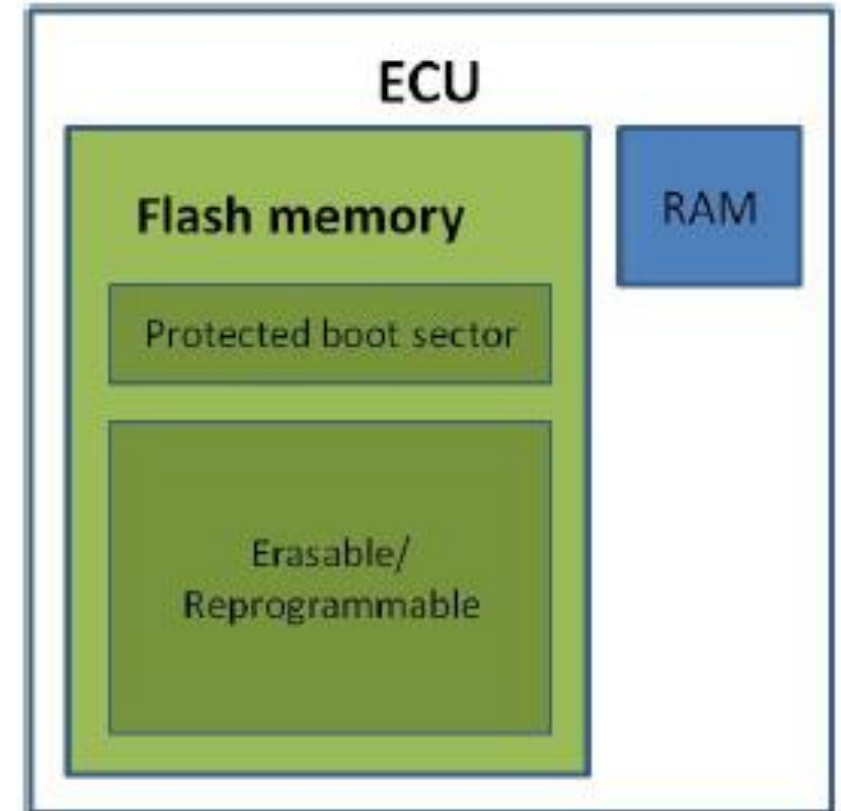
# Format

A standard Intel HEX file format is depicted in Figure 2-6. Figure 2-6: A standard Intel HEX format binary data file. All records start with a colon ":", byte count (Red), address field (Blue), record type (Black), binary data (Green) and a checksum (Yellow) on each record. Figure from SB-Projects [12]. The record types most frequently used are "00" which is a standard data record and "01" which is an end of file record [12].

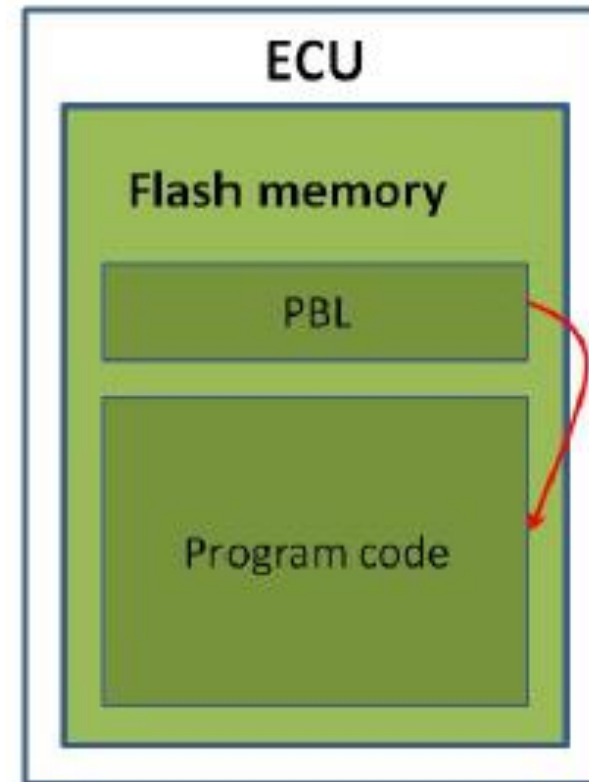
```
:10C00000576F77212044696420796F7520726561CC
:10C010006C6C7920676F207468726F756768206137
:10C020006C6C20746869732074726F75626C652023
:10C03000746F207265616420746869732073747210
:04C040007696E67397
:00000001FF
```

# Memory Layout

- ▶ Flash Memory
- ▶ RAM

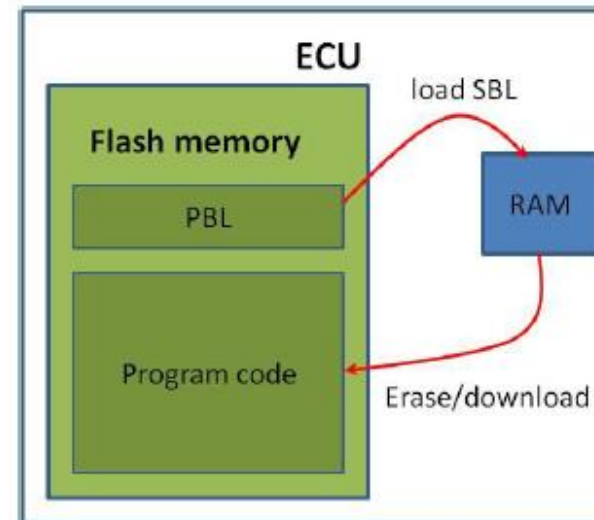


# PBL





# PBL + SBL




## *General negative response codes*

---

When the ECU sends a negative UDS-response to the diagnostic tester (client) the response must include a response code on byte #3 as seen in **Figure 2-17**. There are some general negative response codes or NRC:s which are defined in the UDS-standard in addition to the UDS-service specific NRC:s. The NRC:s are divided into two different byte value ranges depending on the type of errors:

- 0x01-0x7F – Communication related NRC:s.
- 0x80-0xFF – NRC:s which are sent for specific conditions which are

not correct at the time when the request is received by the server.



<b>Byte Value</b>	<b>Negative Response Code (NRC) Definition</b>
0x10	General Reject
0x11	Service Not Supported
0x12	Sub-function Not Supported
0x13	Incorrect Message Length Or Invalid Format
0x22	Conditions Not Correct
0x24	Request Sequence Error
0x31	Request Out Of Range
0x33	Security Access Denied
0x72	General Programming Failure
0x78	Request Correctly Received - Response Pending

NRC : commonly used NRC in the software loading sequence

# NRC(s)

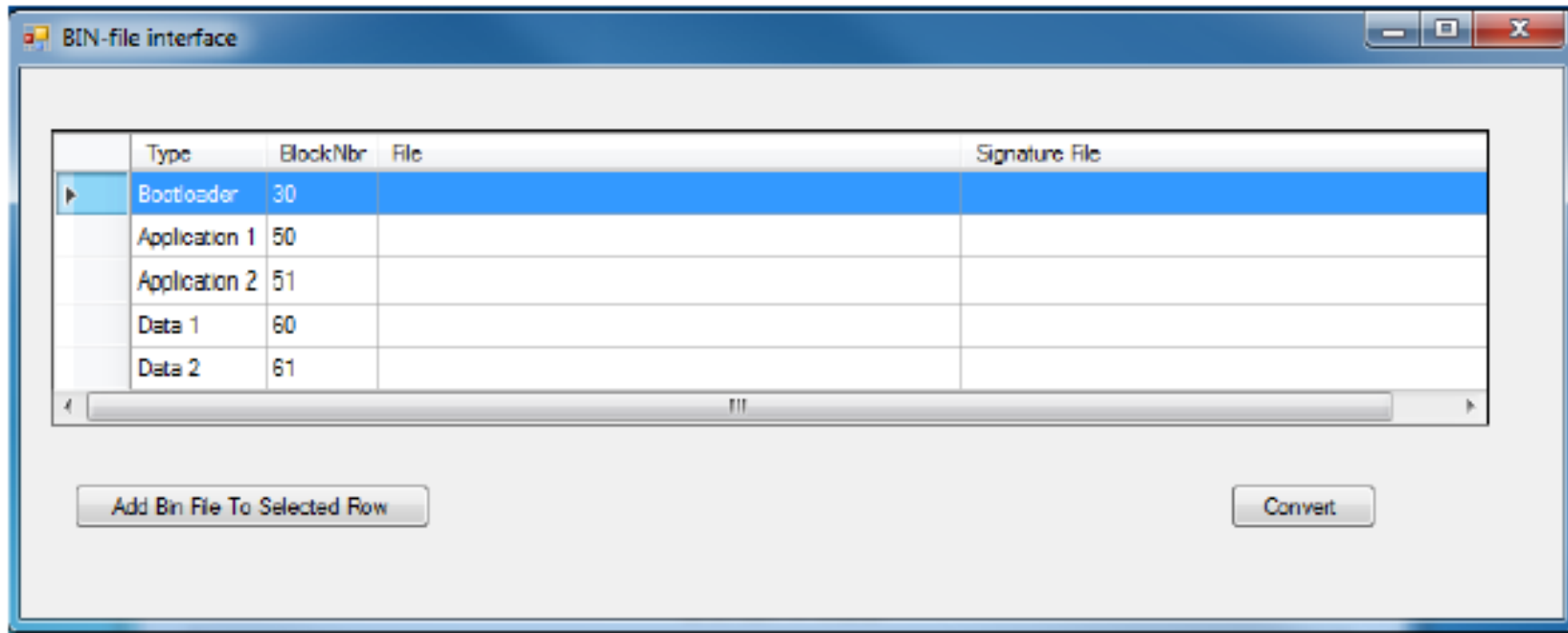
- ▶ General Reject (0x10): Should only be used if none of the NRCs defined in the UDS-standard meet the requirements of the implementation.
- ▶ Service Not Supported (0x11): Should be sent when the ECU receives a UDS-service request with an SID which is unknown or not supported.
- ▶ Sub-function Not Supported (0x12): Should be sent when the ECU receives a UDS-service request with a sub-function which is unknown or not supported.
- ▶ Incorrect Message Length Or Invalid Format (0x13): Should be sent when the length or format of the received request message does not match what is specified by service.

# NRC(s)

- ▶ Conditions Not Correct (0x22): Should be sent when the prerequisite conditions are not correct.
- ▶ Request Sequence Error (0x24): Should be sent when the ECU expected a different sequence of messages than what was sent by the tester.
- ▶ Request Out Of Range (0x31): Should be sent when the tester requests modifying a value which it does not have authority to change.
- ▶ Security Access Denied (0x33): Should be sent when the security requirements of the ECU are not fulfilled for the current request.
- ▶ General Programming Failure (0x72): Should be sent when the ECU has noticed an error while programming or erasing memory.

# NRC(s)

- ▶ Request Correctly Received - Response Pending (0x78): Should be sent when the request is correctly formulated, but the server has not yet completed the required actions. This NRC is supported in all UDS services.



# Download File Sequence

Byte	Description	Byte Value
#1	Response SID	0xXX
#2	data-parameter#1	0xXX
...	...	0xXX
#n	data-parameter#n-1	0xXX

**Figure 2-16:** The structure of a UDS-response message is defined in the UDS-standard, byte #1 is the request SID, the rest of the bytes are data parameters.

## UDS Request Format



# UDS Response Format

Byte	Description	Byte Value
#1	Negative Response SID	0x7F
#2	Request SID	0xXX
#3	ResponseCode	0xXX

**Figure 2-17: A negative UDS-response will have a response SID 0x7F, byte #2 will contain the request SID and byte #3 will contain a response code which contains information why the request was not accepted by the ECU.**

Sub-function Byte Value = 0x81								
Bit #	7	6	5	4	3	2	1	0
Value	1	0	0	0	0	0	0	1

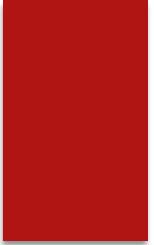
**Figure 2-19: Bit value representation of a sub-function in an UDS-request. Bit #7 is set to 1 to indicate that it is a suppress positive response UDS-request.**

The corresponding diagnostic session control request without suppress positive response can be seen in [Figure 2-20](#).

Sub-function Byte Value = 0x01								
Bit #	7	6	5	4	3	2	1	0
Value	0	0	0	0	0	0	0	1

**Figure 2-20: Bit value representation the SID of a sub-function in an UDS-request.**


# Suppress Positive Response



Byte #	1	2	3	4	5	6	7	8
Value	0x10	0x02	-	-	-	-	-	-

**Figure 2-21: A standard Diagnostic Session Control UDS-request to enter programming session. The request SID 0x10 on byte #1 and sub-function 0x02 on byte #2 according to the general message conventions.**

Diagnostic Session Control : Programming session  
Request



Byte #	1	2	3	4	5	6	7	8
Value	0x50	0x02	0x00	0x0A	0x01	0xF4	-	-


**Figure 2-22: A standard positive Diagnostic Session Control UDS-response. It will return the response SID on byte #1 and the sub-function on byte #2 according to the general message conventions. In addition a positive Diagnostic Session Control service response will contain P2 and P2 extended timing values on bytes #3-6.**

Diagnostic Session Control : Programming session  
Response

**Table 2-3: Negative response codes supported by the Diagnostic Session Control service**

<b>Byte Value</b>	<b>Negative Response Code (NRC) Definition</b>
<b>0x12</b>	<b>Sub-function Not Supported</b>
<b>0x13</b>	<b>Incorrect Message Length Or Invalid Format</b>
<b>0x22</b>	<b>Conditions Not Correct</b>

# Negative Response codes



Byte #	1	2	3	4	5	6	7	8
Value	0x11	0x01	-	-	-	-	-	-

**A standard ECU Reset UDS-request with sub-function Hard Reset.**

The UDS-service ECU Reset is used to perform a reset of the ECU. This service is usually used in post-programming after a reprogramming of an ECU. There are several sub-functions defined in the UDS-standard but only the sub-function “Hard Reset” is used in the software loading sequences used. After ECU Reset has been performed the ECU should enter the default session.

# ECU Reset service: Request

**Table 2-4: Negative response codes supported by the ECU Reset service**

<b>Byte Value</b>	<b>Negative Response Code (NRC) Definition</b>
<b>0x12</b>	<b>Sub-function Not Supported</b>
<b>0x13</b>	<b>Incorrect Message Length Or Invalid Format</b>
<b>0x22</b>	<b>Conditions Not Correct</b>
<b>0x33</b>	<b>Security Access Denied</b>

# ECU Reset service: NRCs

# Security Access Service

---

In order to prevent unauthorized access to the ECU the vehicle manufacturers implement the Security Access service, which is specified in the UDS-standard.

Generally, security access is required before any transfer of new software to the ECU can be performed. The Security Access service utilizes a seed and key structure; the tester (client) will request security access with a UDS-request (see [Figure 2-24](#)).



Byte #	1	2	3	4	5	6	7	8
Value	0x27	0x01	-	-	-	-	-	-

**Figure 2-24: Example of a request seed Security Access UDS-request.**

The ECU will then respond with a positive response, which contains what is called a security seed, the length of the seed is vehicle manufacturer specific but is typically 3-4 bytes long (see [Figure 2-25](#)).

Byte #	1	2	3	4	5	6	7	8
Value	0x67	0x01	0xC6	0xF8	0x98	0x69	-	-


**Figure 2-25: Example of a Security Access UDS-response containing the security seed.**

The security seed will then be used in a vehicle manufacture specific algorithm, which will return a security, key that the tester (client) will send in an UDS-request (see [Figure 2-26](#)).

# Security Access service

# Security Access service


- ▶ If the security key returned by the tester (client) is correct the ECU will
  - ▶ respond with a positive UDS-response (see Figure 2-27). After this the tester will
  - ▶ be granted security access at the requested security level.
- 
- ▶ The vehicle manufacturers have the option of adding different security levels to differentiate
  - ▶ what level of access is given to the user of the security access. Different levels of
  - ▶ security access can be implemented by using different sub-function byte values to
  - ▶ symbolize different levels of security. The relation between the request seed
  - ▶ request and send key request is fixed so that if the request seed byte value is 0x01
  - ▶ then the send key byte value will be 0x02. If the request seed byte value is 0x03
  - ▶ then the send key byte value will be 0x04.



Byte #	1	2	3	4	5	6	7	8
Value	0x27	0x02	0xBF	0xFC	0xE7	0xC3	-	-

Example of a send key Security Access UDS-request containing the calculated security key.

# Security Access service



Byte #	1	2	3	4	5	6	7	8
Value	0x67	0x02	-	-	-	-	-	-

**Figure 2-27: Example of a positive Security Access UDS-response granting security access to the tester (client).**

# Security Access service



# Security Access service

The general negative response codes supported by the Security Access service are listed in [Table 2-5](#) and the Security Access specific NRC:s are listed in [Table 2-6](#).

**Table 2-5: General negative response codes supported by the Security Access service**

Byte Value	Negative Response Code (NRC) Definition
0x12	Sub-function Not Supported
0x13	Incorrect Message Length Or Invalid Format
0x22	Conditions Not Correct
0x24	Request Sequence Error
0x31	Request Out Of Range

**Table 2-6: Negative response codes specific to Security Access service in the software loading sequence**

<b>Byte Value</b>	<b>Negative Response Code (NRC) Definition</b>
<b>0x35</b>	<b>Invalid Key</b>
<b>0x36</b>	<b>Exceeded Number Of Attempts</b>
<b>0x37</b>	<b>Required Time Delay Not Expired</b>

NRC(s)

# NRC(s)

- ▶ Invalid Key (0x35): Sent if the security key sent by the tester (client) does not match the expected key value.
- ▶ Exceeded Number Of Attempts (0x36): Sent if a delay timer is active due to too many incorrect send key Security Access requests.
- ▶ Required Time Delay Not Expired (0x37): Sent if the delay timer is still active.

# Read By Identifier service

- ▶ Service to read data at a memory location specified, used in a flashing sequence to
- ▶ read programming-, fingerprint-data and prepare the ECU for reprogramming. It is
- ▶ a vehicle manufacturer specific step that is sometimes included in the preprogramming part of the software loading sequence.



**Table 2-9: General supported negative response codes in the UDS-service Read By Identifier**

<b>Byte Value</b>	<b>Negative Response Code (NRC) Definition</b>
<b>0x12</b>	<b>Sub-function Not Supported</b>
<b>0x13</b>	<b>Incorrect Message Length Or Invalid Format</b>
<b>0x22</b>	<b>Conditions Not Correct</b>
<b>0x31</b>	<b>Request Out Of Range</b>

Read By Identifier service : NRCs

**Table 2-10: Read By Identifier specific negative response code in the software loading sequence**

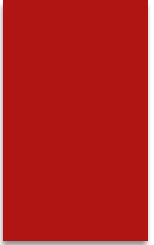
<b>Byte Value</b>	<b>Negative Response Code (NRC) Definition</b>
<b>0x14</b>	<b>Response Too Long</b>

- **Response Too Long (0x14):** Should be sent if the total length of the response exceeds the maximum length defined in the governing transport protocol.

# Read By Identifier service : NRCs

# Write By Identifier service

- ▶ The Write By Identifier service is used to write data to a specific memory location, for example writing programming data and fingerprint data. Typically
- ▶ this is done right before transferring data to the ECU and/or after a successful software loading sequence. An example of a Write By Identifier UDS-request



Byte #	1	2	3	4	5	6	7	8
Value	0x2E	0xF1	0x58	0x15	0x11	0x19	0x02	0x03

**Figure 2-31: A typical Write By Identifier UDS-service request. The data identifier of the memory location which should be written to is specified on byte #2 and 3. The following bytes represent the data record that should be written to that memory location. The data identifier and record length is variable.**

## WriteData By Identifier service : Request

**Table 2-11: Negative response codes supported by the Write By Identifier service**

<b>Byte Value</b>	<b>Negative Response Code (NRC) Definition</b>
<b>0x13</b>	<b>Incorrect Message Length Or Invalid Format</b>
<b>0x22</b>	<b>Conditions Not Correct</b>
<b>0x31</b>	<b>Request Out Of Range</b>
<b>0x33</b>	<b>Security Access Denied</b>
<b>0x72</b>	<b>General Programming Failure</b>

WriteData By Identifier service : NRCs


# Routine Control Service

- ▶ The Routine Control service is one of the most flexible services in the UDS standard.
- ▶ It is typically used in the software loading sequence to check programming pre-conditions and disable failsafe reactions in pre-programming (see Chapter 2.8.4). Also used for performing erasure of EEPROM or flash memory before downloading a new block, checking for valid flash memory and application after the transfer of data. The structure of a routine control service is shown in Figure 2-32.
- ▶ One of the Routine Control services defined in the UDS-standard is the Erase Memory or Erase Flash routine. It will perform the erasure of EEPROM and flash memory before loading a new block. It has the routine identifier FF01, specified on bytes #3-4 in the UDS-request. The remaining bytes of the request contain a routine control option record which is of variable length depending on vehicle manufacturing specification and which routine identifier is used. An example of a start Erase Flash Routine Control service is shown.

# Routine Control Service

Byte	Description	Byte Value
#1	Request SID	0x31
#2	Sub-function	0x01
#3	Routine ID #1	0xFF
#4	Routine ID #2	0xFF
#5	Routine Control Option Record	0xFF
..		..
#n		0xFF

**Figure 2-32: Start Routine Control Service UDS-request. Sub-function byte value 0x01 indicates Start Routine. The routine identifier (ID) is written on bytes #3-4.**



Byte #	1	2	3	4	5	6	7	8
Value	0x31	0x01	0xFF	0x00	0x01	0x15	-	-

**Figure 2-33: A typical Erase Flash Routine Control UDS-request. This request aims to start an erasure of a EEPROM or flash memory sector before downloading block 0x15.**

## Routine Control Service: Request



**Table 2-12: Negative response codes supported by the Routine Control service**

<b>Byte Value</b>	<b>Negative Response Code (NRC) Definition</b>
<b>0x12</b>	<b>Sub-function Not Supported</b>
<b>0x13</b>	<b>Incorrect Message Length Or Invalid Format</b>
<b>0x22</b>	<b>Conditions Not Correct</b>
<b>0x24</b>	<b>Request Sequence Error</b>
<b>0x31</b>	<b>Request Out Of Range</b>
<b>0x33</b>	<b>Security Access Denied</b>
<b>0x72</b>	<b>General Programming Failure</b>

WriteData By Identifier service : NRCs

# Request Download service

- ▶ The Request Download service together with Transfer Data and Request Transfer
- ▶ Exit constitute the main components of the actual transfer of new block data to a reprogrammable ECU. A Request Download UDS-request contains information of
- ▶ which memory address the data block should be downloaded to, how large the
- ▶ data block is in bytes and if the data block is encrypted and/or compressed. The structure of a Request Download UDS-request can be seen.
- ▶ The data format identifier specifies if data is encrypted and/or compressed.
- ▶ The address and length format identifier specifies the size of the memory address
- ▶ and size fields in bytes.
- ▶ The memory address indicates where the data should be
- ▶ written and memory size specifies how many bytes are in the block to be
- ▶ downloaded.

Byte	Description	Byte Value
#1	Request SID	0x34
#2	dataFormatIdentifier	0xFF
#3	addressAndLengthFormatIdentifier	0xFF
#4	memoryAddress field	0xFF
..		..
..	memorySize field	..
#n		0xFF

**Figure 2-34: Structure of a Request Download Service UDS-request.** The data format identifier specifies if data is encrypted and/or compressed. The address and length format identifier specifies the size of the memory address and size fields in bytes. The memory address indicates where the data should be written and the memory size field specifies how many bytes are in the block to be downloaded.

# Request Download

Byte #	1	2	3	4	5	6	7	8
Value	0x34	0x00	0x41	0x31	0x00	0x00	0x05	0x34

**Figure 2-35:** An example of a Request Download UDS-request with address and length identifier on byte #3 with a byte value 0x41. The 4 represented by bits #7-4 on byte #3 indicate that the memory size is written on 4 bytes. The 1 represented by bits #3-0 on byte #3 indicates that the memory address field is written on 1 byte. Bytes #5-8 indicate how large data block should be downloaded, in this case 1332 bytes (0x0534).


Byte #	1	2	3	4	5	6	7	8	9	10	11
Value	0x34	0x00	0x44	0x00	0xF1	0x00	0x00	0x00	0x00	0x05	0x34

**Figure 2-36:** An example of a Request Download UDS-request with address and length identifier on byte #3 with a byte value 0x44. The 4 represented by bits #7-4 on byte #3 indicate that the memory size is written on 4 bytes. The 4 represented by bits #3-0 on byte #3 indicate that the memory address field is written on 1 byte. Bytes #8-11 indicate how large data block should be downloaded, in this case 1332 bytes (0x0534).

# Request Download

# Request Download: Response

- ▶ The response from a Request Download UDS-request contains information
  - ▶ about how large Transfer Data requests are accepted by the ECU. The UDSstandard
  - ▶ requires that the ECU must be able to accept at least a Transfer Data
  - ▶ request of the length specified by the “MaxNumberOfBlockLength” contained in
  - ▶ the response. Transfer Data requests of shorter length than what is indicated by the
  - ▶ “MaxNumberOfBlockLength” are vehicle manufacturer specific if they accept or
  - ▶ not. However, the last Transfer Data message may contain less than the maximum
  - ▶ size of data packages, which must be handled by the ECU according to the UDSstandard.



Byte #	1	2	3	4	5	6	7	8
Value	0x74	0x20	0x0C	0x62	-	-	-	-

**Figure 2-37: An example of a Request Download UDS-response. On byte #2 the length format identifier indicates on how many bytes the “MaxNumberOfBlockLength” should occupy. The 2 represented by bits #7-4 on byte #2 indicates that the “MaxNumberOfBlockLength” will be written on two bytes. On byte #3-4 the “MaxNumberOfBlockLength” is equal to 3170 (0x0C62) in this case. This value includes the block sequence counter and request SID of the Transfer Data Service.**

# Request Download: Response

Table 2-13: The negative response codes supported by the Request Download service

Byte Value	Negative Response Code (NRC) Definition
0x12	Sub-function Not Supported
0x13	Incorrect Message Length Or Invalid Format
0x22	Conditions Not Correct
0x24	Request Sequence Error
0x31	Request Out Of Range
0x33	Security Access Denied
0x72	General Programming Failure

- **Request Out Of Range (0x31)** – Should be sent out if any of the data fields in [Figure 2-34](#) are not valid, i.e. dataformatIdentifier, addressAndLengthFormatIdentifier, memory address and size fields are invalid (see [Figure 2-34](#)).

# Request Download: NRCs

# Transfer Data (0x36)

## *Transfer Data service*

The data in the software loading sequence is sent out in Transfer Data service-requests. After the tester has received a positive Request Download response the client (tester) will be permitted to send Transfer Data requests.

The Transfer Data request contains a block sequence counter, which can be used in error handling during the data transfer. The block sequence counter should start at 0x01, and then add 1 (one) every new Transfer Data request and when it reaches 0xFF it should roll over to 0x00. If two Transfer Data requests received by the ECU contain the same block sequence counter the ECU should send positive responses on both. It is recommended that the ECU only write the data to the flash memory from the first Transfer Data request with the same block sequence counter. A typical Transfer Data request sequence is shown in [Figure 2-38](#). This example shows the transfer of a block of size 1076 bytes. This ECU has a “MaxNumberBlockLength” of 514 bytes including the request SID and block sequence counter. This means that 512 bytes will be sent in the first two Transfer Data requests and the last request will contain the last 52 bytes in the block file.



## Transfer Data - 0x36

Byte #	1	2	3	Data bytes..			MaxNumberOfBlockLength
Value	0x36	0x01	0xFF	0xFF	.....	0xFF	0xFF

Byte #	1	2	3	Data bytes..			MaxNumberOfBlockLength
Value	0x36	0x02	0xFF	0xFF	.....	0xFF	0xFF

Byte #	1	2	3	Data bytes..			54
Value	0x36	0x03	0xFF	0xFF	.....	0xFF	0xFF

**Figure 2-38: A typical Transfer Data request sequence. This example shows the transfer of 1076 bytes. The ECU has a “MaxNumberBlockLength” of 514 bytes including the request SID and block sequence counter. This means that 512 bytes will be sent in the first two Transfer Data requests and the last request will contain the last 52 bytes in order to complete the sending of 1076 data bytes.**

## Transfer Data - 0x36

**Table 2-14: General supported negative response codes in the UDS-service Transfer Data**

<b>Byte Value</b>	<b>Negative Response Code (NRC) Definition</b>
<b>0x13</b>	<b>Incorrect Message Length Or Invalid Format</b>
<b>0x24</b>	<b>Request Sequence Error</b>
<b>0x31</b>	<b>Request Out Of Range</b>
<b>0x72</b>	<b>General Programming Failure</b>

**Table 2-15: Transfer Data specific negative response codes in the software loading sequence**

<b>Byte Value</b>	<b>Negative Response Code (NRC) Definition</b>
<b>0x71</b>	<b>Transfer Data Suspended</b>
<b>0x73</b>	<b>Wrong Block Sequence Counter</b>
<b>0x92/0x93</b>	<b>Voltage Too High / Voltage Too Low</b>

# Transfer Data - 0x36

## **Transfer Data Suspended (0x71):**

If the memorySize (see Figure 2-34) parameter in the Request Download request does not match the number of bytes sent by the Transfer Data requests this NRC should be sent by the ECU.

- **Wrong Block Sequence Counter (0x73):**

If the ECU notices an error in the “block Sequence Counter” sequence then it should send this NRC.

- Voltage Too High / Voltage Too Low (0x92/0x93):

Should be sent if the voltage measured by the ECU are outside of the permitted range for a software loading sequence

# Transfer Exit (0x37)

## ***Request Transfer Exit service***

The Request Transfer Exit UDS-request should be sent after all the data has been transferred with the Transfer Data requests. A typical Request Transfer Exit UDS-request in the software loading is shown in [Figure 2-39](#).

Byte #	1	2	3	4	5	6	7	8
Value	0x37	-	-	-	-	-	-	-

**Figure 2-39: A typical Request Transfer Exit UDS-request in the software loading.**

# Transfer Exit (0x37)

The format of the Request Transfer Exit response is vehicle manufacture specific; some of BW-PDS customers will send the checksum for the downloaded block in the response (see [Figure 2-40](#)). In these cases it is the responsibility of the tester (client) to check that this checksum matches and if it does not the tester should abort the software loading sequence to avoid loading the incorrect data onto the ECU. Other manufacturers use a separate Routine Control in which the tester sends the checksum to the ECU and the ECU will handle the possible checksum error instead.

Byte #	1	2	3	4	5	6	7	8
Value	0x77	0xFF	0x7A	-	-	-	-	-

**Figure 2-40: Request Transfer Exit response with a block checksum included in the response.**

# Transfer Exit (0x37)

The negative response codes supported by the Request Transfer Exit are listed in [Table 2-16](#).

**Table 2-16: Negative response codes supported by the Request Transfer Exit service**

<b>Byte Value</b>	<b>Negative Response Code (NRC) Definition</b>
<b>0x13</b>	<b>Incorrect Message Length Or Invalid Format</b>
<b>0x24</b>	<b>Request Sequence Error</b>
<b>0x31</b>	<b>Request Out Of Range</b>
<b>0x72</b>	<b>General Programming Failure</b>

Questions?

# Service: Request Download – 0x34

---

1. Compression format support
2. Address / Length Specifier can be used

Tester :

0x10	0x09	0x34	AA	AA	AA	AA	BB
------	------	------	----	----	----	----	----

ECU :

0x30	0x00	0x00	Pad	Pad	Pad	Pad	Pad
------	------	------	-----	-----	-----	-----	-----

Tester :

0x21	0xBB	0xBB	0xBB	Pad	Pad	Pad	Pad
------	------	------	------	-----	-----	-----	-----

ECU :

0x04	0x74	0xCC	0xCC	Pad	Pad	Pad	Pad
------	------	------	------	-----	-----	-----	-----



## Service: Data Transfer – 0x36

---

0x02	0x36	0x01					
0x7F	0x10	0x22					
0x7F	0x10	0x22					
0x7F	0x10	0x22					
0x7F	0x10	0x22					

## Service: Transfer Exit – 0x37

---

0x7F	0x10	0x22					
0x7F	0x10	0x22					
0x7F	0x10	0x22					
0x7F	0x10	0x22					
0x7F	0x10	0x22					